

# Imagina: A Cognitive Abstraction Approach to Sketch-Based Image Retrieval

by

Manolis Kamvysselis and Ovidiu Marina

Submitted to the Department of Electrical Engineering and Computer Science  
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999

© Manolis Kamvysselis and Ovidiu Marina, MCMXCIX. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper  
and electronic copies of this thesis document in whole or in part.

Author.....  
Department of Electrical Engineering and Computer Science  
May 18, 1999

Certified by .....  
Patrick H. Winston  
Ford Professor of Artificial Intelligence and Computer Science  
Thesis Supervisor

Accepted by .....  
Arthur C. Smith  
Chairman, Department Committee on Graduate Students

# **Imagina: A Cognitive Abstraction Approach to Sketch-Based Image Retrieval**

by

Manolis Kamvysselis and Ovidiu Marina

Submitted to the Department of Electrical Engineering and Computer Science  
on May 18, 1999, in partial fulfillment of the  
requirements for the degrees of  
Bachelor of Science in Electrical Engineering and Computer Science  
and  
Master of Engineering in Electrical Engineering and Computer Science

## **Abstract**

As digital media become more popular, corporations and individuals gather an increasingly large number of digital images. As a collection grows to more than a few hundred images, the need for search becomes crucial. This thesis is addressing the problem of retrieving from a small database a particular image previously seen by the user. This thesis combines current findings in cognitive science with the knowledge of previous image retrieval systems to present a novel approach to content based image retrieval and indexing. We focus on algorithms which abstract away information from images in the same terms that a viewer abstracts information from an image. The focus in Imagina is on the matching of regions, instead of the matching of global measures. Multiple representations, focusing on shape and color, are used for every region. The matches of individual regions are combined using a saliency metric that accounts for differences in the distributions of metrics. Region matching along with configuration determines the overall match between a query and an image.

Thesis Supervisor: Patrick H. Winston

Title: Ford Professor of Artificial Intelligence and Computer Science

## Acknowledgments

Thanks to all our friends for letting us finish the thesis. *Patrycja, Maria, Gwenaelle, Anna, Sofy, Stephanie, Sabrina, Megan, Natalie, Minnie, Jodi, Crista, Delphine, Michele, Rachel, Caroline, Cami, Lisa, Marion, Nicky, Carolina, Nicole, Anthi, Georgia, Susan, Deirdre, Agneta, Angela, Aletia, Angelita, Barby, Maggy, Norma, Chelsea, Sarah, Adrianne, Allison, Eve, Tracey, Tammy, Ania, Hala, Rania, Mary, Rebecca, Tanya, Maya, Alex, Agnes, Orsi, Karen, Xilonin, Paula, Stacey*, you know who you are.

To our supervisor, *Patrick*, who always supported us with direct and constructive comments, we owe our eternal gratitude. His kind words and most elaborate description of our project will remain unforgettable to us:

Imagina nourishes the extended literature, leading elaborate concepts to ulterior academic labor. Manoli and Strider, two uplifting riots, bring alive the institute's overworked nerds.

And lastly, to *Anthony*, thanks for putting up with us.

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Design of Imagina as an Image Retrieval System . . . . .	11
1.2	A Sketch Query allows greater user versatility . . . . .	12
1.3	A cognitive approach should find the image closest to the user's expectations . . . .	12
1.4	Multiple representations allow handling a large variety of inputs . . . . .	13
1.5	Multiple abstraction levels allow noise tolerance and early pruning . . . . .	14
1.6	Region-based searching enables user input . . . . .	15
1.7	Making the representation explicit keeps no secrets from the user . . . . .	16
1.8	Summary . . . . .	16
1.9	Organization of the thesis . . . . .	17
<b>2</b>	<b>Query Systems</b>	<b>18</b>
2.1	An Overview of Previous Systems . . . . .	19
2.2	Image Retrieval Systems . . . . .	20
2.2.1	Query by Image Content (QBIC) . . . . .	20
2.2.2	The Virage Engine of the AltaVista Picture Finder . . . . .	20
2.2.3	WISE: A Wavelet-based Image Search Engine . . . . .	21
2.2.4	Photobook . . . . .	21
2.2.5	VisualSEEk . . . . .	22
2.2.6	Blobworld . . . . .	23
2.3	Imagina in Comparison . . . . .	23
<b>3</b>	<b>Theoretical Foundations</b>	<b>25</b>
3.1	Streams and Counter-Streams . . . . .	25

3.2	Gestalt Psychology . . . . .	26
3.3	Canonical Views . . . . .	27
3.4	The Limitations of Passive Input . . . . .	28
3.5	Layering of Processing . . . . .	29
3.6	Combining Unrelated Measures . . . . .	29
<b>4</b>	<b>Color</b>	<b>31</b>
4.1	The Red-Green-Blue (RGB) Color Space . . . . .	31
4.2	The Hue-Saturation-Value (HSV) Color Space . . . . .	32
4.2.1	Computing the HSV Representation . . . . .	33
4.2.2	The HSV Distance Metric . . . . .	35
4.3	Other Color Spaces . . . . .	38
4.4	Use of Color Spaces in Imagina . . . . .	40
4.5	Color-Based Indexing . . . . .	40
4.6	Comparison Based on Color Histograms . . . . .	41
4.6.1	Discussion of Previous Approaches . . . . .	41
4.6.2	Color Histogram Definition in Imagina . . . . .	42
4.6.3	Color Histogram Matching . . . . .	43
4.7	Comparison Based on Color Mappings . . . . .	46
4.8	Uses of Color-Based Comparisons . . . . .	51
<b>5</b>	<b>Filters</b>	<b>52</b>
5.1	Opponency of Values as a System of Measurement . . . . .	53
5.2	Types of Measurement . . . . .	54
5.3	Types of Filters . . . . .	54
5.3.1	Spot Filters . . . . .	55
5.3.2	Edge Filters . . . . .	55
5.3.3	Excitatory and Inhibitory Filtering Field Shapes . . . . .	57
5.4	Output Modifications . . . . .	57
5.5	Difficulty of Filter Application . . . . .	58
<b>6</b>	<b>Image Segmentation</b>	<b>59</b>
6.1	The <i>Grow</i> Algorithm . . . . .	60

6.2	The <i>Adaptive Grow</i> Algorithm . . . . .	62
6.3	The <i>Progressive Grow</i> Algorithm . . . . .	62
6.4	Methods for General Segmentation . . . . .	64
6.4.1	Using Saliency for Segmentation . . . . .	64
6.4.2	Using Hierarchy for Segmentation . . . . .	67
6.5	A Hierarchical Recursive Adaptive Region Segmentation Algorithm . . . . .	68
6.5.1	The Algorithm as a Feedback Approximation . . . . .	69
6.5.2	Extensions . . . . .	71
<b>7</b>	<b>Boundary Edge Extraction</b>	<b>72</b>
7.1	Prior work . . . . .	73
7.1.1	The <i>Crust</i> Algorithm . . . . .	73
7.2	Pixel-Based Boundary Extraction . . . . .	74
7.3	Window-Based Boundary Extraction . . . . .	76
7.3.1	Algorithm Overview . . . . .	76
7.3.2	Serialized vs. Parallel Algorithm . . . . .	77
7.3.3	Computation within a Window . . . . .	77
7.3.4	Determining a tangent . . . . .	82
7.3.5	Post-processing on the Edges Found . . . . .	85
7.3.6	Improvements and Extensions . . . . .	87
7.4	Conclusions . . . . .	89
<b>8</b>	<b>Shape Representation and Region Matching</b>	<b>90</b>
8.1	Literature on Shape Descriptors . . . . .	90
8.1.1	Image Retrieval and Shape Description . . . . .	91
8.1.2	Vision and Shape Description . . . . .	91
8.1.3	Graphics and Shape Description . . . . .	91
8.2	Shape Descriptors Used in Imagina . . . . .	92
8.3	Volume Based Representation and Comparison . . . . .	94
8.3.1	Approximating the Inside Points . . . . .	94
8.3.2	Scaling and Alignment . . . . .	94
8.3.3	Similarity Metric . . . . .	95

8.3.4	Results . . . . .	95
8.3.5	Shortcomings and Strong Points of a Volume Based Representation . . . . .	97
8.4	Segment-Based Representation . . . . .	98
8.5	Angle Based Representation and the Extended Gaussian Image . . . . .	100
8.6	Shape Descriptor based on Protrusions . . . . .	102
8.7	A metric for combining comparisons from individual methods . . . . .	102
8.8	Conclusion . . . . .	104
<b>9</b>	<b>Image Retrieval and Matching</b>	<b>105</b>
9.1	Challenge of Classification . . . . .	106
9.2	Combining the Representations . . . . .	107
9.3	Dealing with Multiple Regions within an Image . . . . .	107
9.4	Image Matching . . . . .	108
9.4.1	Comparing Configurations Given a Region Mapping . . . . .	110
9.4.2	Matching Issues . . . . .	110
<b>10</b>	<b>Extensions</b>	<b>112</b>
10.1	Preprocessing: Improving Image Fidelity . . . . .	113
10.2	Database: Principles of a Large Database Design . . . . .	114
10.3	Database: Representation of Images . . . . .	114
10.4	Database: Structuring a Large Database . . . . .	116
10.5	Database: Abstraction and Specification Functions . . . . .	116
10.5.1	Abstraction Functions . . . . .	117
10.5.2	Different Levels of Abstraction . . . . .	117
10.5.3	Specification functions . . . . .	117
10.5.4	Coding Abstraction functions . . . . .	118
10.6	Learning: Using Feedback to Improve Search Performance . . . . .	118
10.6.1	Re-Weighting Functions . . . . .	119
10.6.2	Speed up questions . . . . .	119
10.7	Module Extensions: Parallel Implementations . . . . .	119
10.8	Additional Modules: Texture and Shadows . . . . .	120
<b>11</b>	<b>Conclusion</b>	<b>121</b>

11.1	Contributions . . . . .	121
11.1.1	Sketch Based Drawing . . . . .	121
11.1.2	Global Segmentation based on Color . . . . .	122
11.1.3	Matching based on region shape . . . . .	122
11.1.4	Multiple Levels of Detail . . . . .	122
11.1.5	Explicitness of representation . . . . .	123
11.2	Shortcomings . . . . .	123
11.3	Conclusion . . . . .	123
<b>A</b>	<b>The Imagina System</b>	<b>124</b>
A.1	Implementation of Imagina as a Tool . . . . .	124
A.2	Sketch and Image Upload . . . . .	125
A.3	No Computational Bottlenecks . . . . .	126
A.4	Extensible Architecture . . . . .	126
A.5	Easily replaced algorithmic modules . . . . .	126
A.6	Explicitness of representation . . . . .	126
A.7	Platform Independence . . . . .	127
<b>B</b>	<b>The Algorithm Toolbox</b>	<b>128</b>
<b>C</b>	<b>Altavista's Image Finder</b>	<b>129</b>
<b>D</b>	<b>Image User Queries and Matches</b>	<b>133</b>
<b>E</b>	<b>Image User Queries and Matches</b>	<b>145</b>



# Chapter 1

## Introduction

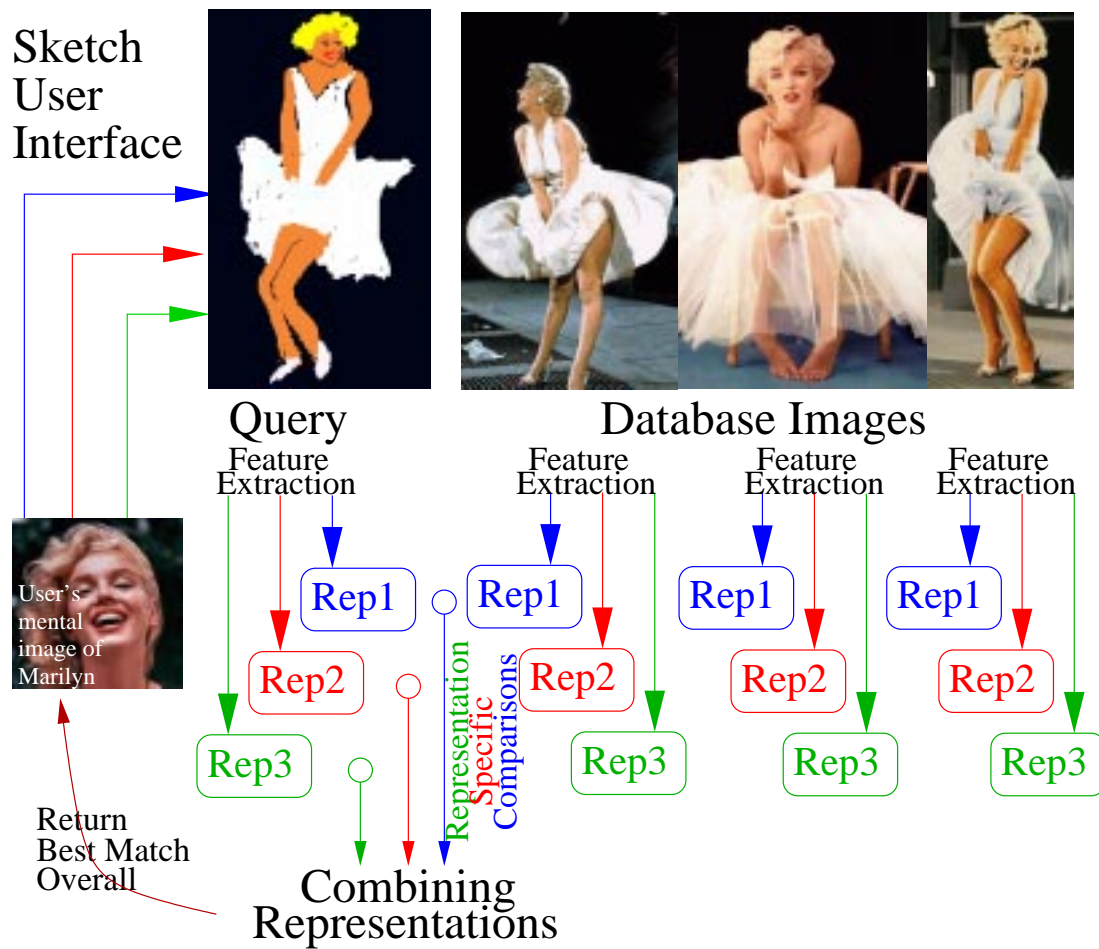
*He that will not sail till all dangers are over must never put to sea.*

Thomas Fuller

As digital media become more popular, corporations and individuals gather an increasingly large number of digital images. As a collection grows to more than a few hundred images, the need for search becomes crucial. This thesis is addressing the problem of retrieving from a small database a particular image previously seen by the user. This situation is only too real for some of us who have large image collections in digital form. This thesis grew out of our need for a tool to help us manage our own growing image galleries.

We created Imagina, a system that retrieves images from a small database based on image content. The user queries the system by either drawing a sketch of the desired image or by specifying the Internet address of an image similar to the one sought.

Many existing systems do or claim to do image retrieval by image content. Most of these systems however address only one part of a complex problem. Many of the questions that lay the foundations of image retrieval remain unanswered. Our goal in constructing Imagina is to provide new insights on these basic questions, rather than to provide a definitive solution to the problem of image retrieval. Along these lines, we feel our contributions to be in the areas of color-based segmentation, edge extraction and shape representations for recognition and matching.



Imagina System Overview: The user starts with an abstract mental image of the object they would like to retrieve. This image is made concrete by drawing a sketch. Features are extracted from the sketch, constructing multiple representations of the query. Specific similarity functions compare each representation of the query to representations previously extracted from the database images. The different similarity metrics are combined and the best matches overall are returned to the user along with visual information on how each match occurred. The user can then use this information to formulate a more precise query.

## 1.1 Design of Imagina as an Image Retrieval System

The combination of the following aspects of Imagina sets it apart from other image retrieval tools:

- **User input is in form of a sketch**, which allows greater versatility in the query. Ease of use is further increased by allowing the user to query based on any image that can be downloaded from the Internet.
- **We use a cognitive approach in designing our algorithms**. The main characteristics of such an approach is having only few processing steps, even if each step involves massively parallel computation.
- **Many independent simple methods** are used in the matching, rather than a single complex one. This allows the system to work in a variety of image domains rather than concentrating in features easy to detect in a particular image set.
- **Multiple levels of abstraction** are used within each method allowing first to match descriptions at a coarse resolution and then to narrow the search with finer-grain comparisons. This approach also allows a much greater robustness against noise present in either the database images or the drawn sketch.
- **A region-based image analysis** is used, instead of image indexing based on global metrics. This enables a more natural user interaction, emphasizing image objects rather than global values that may be hard for a user to understand or reproduce.
- **The representation is made explicit** by graphically displaying the criteria used in a match. This allows a constant check of the credibility of the system's results for the programmer. It also allows the user to better understand how the sketch drawn is interpreted.

## **1.2 A Sketch Query allows greater user versatility**

The capability of searching a database of images has become as crucial and should become as natural as text search has become for text databases. The medium supporting the search should be the same as the medium of the documents to be retrieved. Text-based searching of images is therefore unfit. Searching for “three people standing in front of a truck” would require that the database maintainer has pre-indexed every image for each of its elements, including actions such as “standing” and spatial relations such as “in front of”. Alternatively, if the indexing process is to be automated, the system should either be able to recognize any type of object in a database to be indexed and know a name for it, or create mental representations of words in the query. Neither of these questions has been solved yet.

Based on the above discussion, content-based retrieval cannot and should not be done based on a text interface, unless computer vision research is able to reliably construct image models from either text or images. Therefore, the user input should be an image itself. Alternatively, it might be simpler to let the user choose from a set of images already in the database the ones that more closely match the image to be retrieved. This would allow precomputation on these images and feature extraction off-line, thus speeding up the query time, but unfortunately limiting the user’s versatility. If full freedom is to be left to the user, the only practicable medium for a query input is a sketch.

## **1.3 A cognitive approach should find the image closest to the user’s expectations**

Of course, the sketch interface is simply a way of letting the user express their mental picture of the desired image. The ultimate goal of Imagina is to find the image closest to the mental picture of the user. Therefore, in the transition from the sketch into a representation, an effort should be made to keep the representations as close as possible to those of the user. Images will be matched to the sketch using abstractions from both sketch and database images. If the abstractions used are similar or to those of the user, then the match will be closest to the user’s expectations.

We will therefore use findings on how humans process images, in order to replicate computationally the biological functions involved in visual processing. If cognitive criteria are used in determining a metric for similarity, the match returned will be more likely to resemble the one ex-

pected by the user. In two simple rules to follow, the requirement for biological feasibility guides us towards algorithms that on one hand only require only a few steps with parts that can be executed in parallel, and on the other hand have been shown reachable within the limits of processing our visual system can do.

This is not always straightforward however, because of a current lack of understanding of the intricacies of the visual computations performed by human brains. Moreover the limitations of serialized processing imposed by the common computational architectures will force us to implement serial versions of the devised parallelizable algorithms. We should make sure that none of the assumptions we make about the algorithms we use has an intrinsic serializability requirement.

## **1.4 Multiple representations allow handling a large variety of inputs**

The best match for a given input can be determined in two ways. Either a single measure is used, which has to be evaluated so exactly that indexing is accurate, or several less accurate measures are used, which when put together yield a more accurate and robust measure.[39] It is usually much cheaper to find an approximate answer than an exact answer, and usually even several approximations are far cheaper than one exact result. Therefore the obvious solution is to come up with several indexing processes, and use their combined output for retrieval. Such an approach is true for biological systems, which often have multiple parallel processing streams. Moreover, it is becoming more and more common in computational approaches.

Inspired by this model, interpretation and comparison of images and regions in Imagina is not done by hard-wired algorithms that are finely tuned to the picture domain we are processing. Instead, a multitude of representations is used for each image, providing many approximate representations of the data. This allows for a graceful degradation of performance on unexpected input, rather than a total collapse. Like biological systems, Imagina is therefore able to deal with unusual situations as well as the ones it was designed for. Using multiple representations of an image is an easy way to achieve such an independence from problem specificity.

The goal of a system which performs image processing is to use some, or all, of the sources of information available in an image in order to come close to human visual processing in performing indexing and searching. Unfortunately, in segmenting an image, only color is used, even though it has been shown that the human eye is also using other metrics such as texture, shadows, and

motion. A more robust segmentation could have been achieved if more than one metric had been used. Motion is out of the question, since we are only dealing with static images, but texture could have been used. Our rationale in ignoring other metrics, is that a user will be less likely to draw a query with texture and shadowing. The effort that would have been put into detecting and storing those would have only paid off in the segmentation stage, and not in the recognition stage, since recognition of a sketch most rely most of all on shape, other metrics being only imprecisely input to the system. Fortunately, our work on color has provided good algorithms for segmenting the image into regions, compensating for the lack of alternative methods.

## 1.5 Multiple abstraction levels allow noise tolerance and early pruning

Another characteristic of Imagina that was inspired from the biological world is the multiple abstraction levels at which a match can occur. Current knowledge of cognitive processes that occur in recognition exhibit two opposite streams in the human visual system. One abstracts from the current view into more general objects, the other one makes the remembered abstract objects more specific to match the current image perceived. A match is the intersection of the two streams, between the abstract objects being remembered and the concrete image being perceived.

A similar hierarchy of abstractions is found in Imagina. We use the term abstraction to describe one particular representation of an image, containing usually partial information at a particular scale. Abstractions are obtained by concentrating on a particular feature, such as color, orientation, or shape, as the previous paragraph outlined. Moreover, these abstractions are applied at different levels.

These levels can be thought of as different resolutions at which the image is observed. The lowest level abstractions are providing more detail, and abstractions at higher levels provide greater noise tolerance. The chosen level of detail determines the granularity at which two values in the feature space will be considered separate. This level of detail metric however means very different transformations on the input image as will be detailed in the following chapters.

In this hierarchy of abstractions, the branching of the tree comes from the different methods applied to each image, and each level of the hierarchy comes from these methods being applied at different scales. When a sketch query is input to the system, the sketch is progressively abstracted and the abstractions of the saved images are progressively pruned and made more specific to match

the input sketch. This analogy to the human visual system will be further discussed in section 3.1 on page 25. The benefit of this analysis in the search process is that not all images have to be searched extensively. When a mismatch occurs, the processing for that particular stream can be stopped early. Similarly, only those methods which yield satisfying results need be applied at higher resolutions for a particular image.

The other related benefit to having methods apply at different resolutions is a higher tolerance to noise. Each database image is segmented automatically, without user intervention, and thus each region will contain noise on its boundary. By constructing representations at larger window sizes, the high frequency noise in the data can be filtered out, and the more relevant parts of a region's shape can be compared. Such window sizes apply to every part of the system. Such a uniform design allows for the entire image processing pipeline to be replicated at different resolutions, some more noise tolerant, and others more attentive to details.

## **1.6 Region-based searching enables user input**

In general, users remember one or more particular objects in an image, and they want to retrieve the image that contains them. However, most image search engines represent images in terms of their global features, ignoring the local variations which are relevant to the user. Global features allow a speedup of indexing which has to be done in real time on a large number of images. The simplest approach to this is to have very compact image descriptions which throw away a lot of the information. Unfortunately, much of the information thrown away is relevant to the user. Such systems hope that the remaining information, coupled with repeated searches, will eventually yield an acceptable result.

In order to allow a more comprehensive image indexing, objects would have to be segmented from the image reliably. This, however is an unsolved problem. Also, we have reason to believe it is computationally intensive, since billions of neurons are used in at least some aspect of visual processing in the human brain. In a computer system, we can at most hope that the segmentation will lead to coherent regions that are a good approximation to the objects.

This type of image search can be set up by having local features encoded without regards to the particular object they might represent, and attempting to match them to the user queries individually. Although the objects themselves remain unknown, if the same deterministic process is used on two similar images, the result would be closer than if a third, different image were used.

The main expected difficulty is that regions segmented by the search system may not always be the region which a user expects to search by. This problem can be alleviated by extending the user input to include meta-data on the regions selected. This meta-data can describe which regions should remain connected, which regions are most important in the matching. In the extreme case the user can specify even what features should be matched more importantly between color, angle, shape of regions. Alternatively, various degrees of this meta-data can be learned from the user interaction. Region-based indexing enables this type of object-based user interaction.

## **1.7 Making the representation explicit keeps no secrets from the user**

Most image retrieval systems return simply a similarity value between different images, and even the designers of the system have difficulty interpreting some of the results. In Imagina, comparison can display visually why two regions were matched thus making the assumptions of the system explicit.

In fact, the very possibility of providing visual feedback to the user about the matching results from the fact that shape rather than global features is used in matching regions. A system based on global features can at most feedback a color histogram or a set of patterns extracted, both of which are not intuitive to the user. On the contrary, the shape can be immediately verified or disproved and the representation of the system is thus made explicit.

## **1.8 Summary**

In summary, as an image retrieval system, Imagina has been inspired heavily by the cognitive processes of visual recognition in biological systems. This yields an architecture where matching is done by various parallel methods operating at different levels of abstraction. We summarize this architecture by the notion of *Cognitive Abstraction*: comparison and matching occur at different levels of abstraction, using a variety of cognitive processes running in parallel.



## 1.9 Organization of the thesis

- Chapter 1 gave an overview of the system along with the rationale behind the design choices made.
- Chapter 2 presents an overview of previous work in image retrieval systems and places Imagina within the design space of current image retrieval systems.
- Chapter 3 presents the theoretical foundations of our work within the cognitive science and computer vision literature.
- Chapter 4 presents background work on different color spaces and how we use color to determine similarity at both the pixel and the image levels.
- Chapter 5 describes different filters that can be used to separate regions within an image both by labelling regions and determining boundaries between them.
- Chapter 6 describes two approaches to image segmentation: growing uniform regions and global color-based segmentation.
- Chapter 7 describes in detail how we go from a pixel-based representation of a region to the shape description of its boundary.
- Chapter 8 outlines several shape representations that can be used to compare image regions.
- Chapter 9 addresses the retrieval of regions and images by combining metrics of region similarity and spatial configuration similarity.
- Chapter 10 presents possible future extensions to the work presented in this thesis.
- Chapter 11 provides an overview of the contributions and shortcomings of this thesis.

## Chapter 2

# Query Systems

*Real knowledge is to know the extent of one's ignorance.*

Confucius

Vision is the most important sense that we have. Belying its complexity, visual interaction with the world is straightforward and seamless. Objects in a scene can be recognized almost instantaneously, scenes can be recognized in a fraction of a second, and the memory of something seen can last a lifetime. Computer scientists have tried to emulate these capabilities with computational methods, resulting in only limited success.

Although the human visual system is very remarkable, it can also be very unreliable. Images which are remembered cannot be placed temporally or spatially, objects which are recalled were never really there, and sometimes different scenes are confused and combined in memory. This problem is less noticeable when we recall visual scenes or locations with which we have interacted, rather than when the recall of static images is required. This latter case is becoming more common as the storage and retrieval of large quantities of visual data has been increasing due to the higher storage and processing capacities of modern computer systems.

As a result, a system which can be used as an aid for searching among a large number of images becomes desirable. Content-based image indexing has been proposed and implemented in several systems to date. Some of these have been made available commercially, while most are

still only research projects. These systems rely primarily on color and texture information. Only a few attempts have been made at using configuration and shape description. Although most of these systems perform acceptably well for generic searches, they run into the same difficulties that general text-based search engines encounter.

The problem is that images are noisy, the search vocabulary supplied is very non-descriptive, and the rules of image composition are complex and ill-understood. Thus, although text-based search could be built based on phrase structure, and translation between languages is a feasible exercise nowadays, this is not yet possible for images, because the structure of images in terms of their human interpretation is not understood well enough. Image matching, and thus the translation of an image into another equally-meaningful image is currently on the border between difficult and impossible.

## **2.1 An Overview of Previous Systems**

Systems similar to Imagina have been proposed and implemented before, with varying degrees of success. In order of seniority, these systems have been based on comparisons using color, texture, configuration and shape. Most systems rely primarily on color segmentation, with other processing being used for added reliability and for narrowing down the search space.

Texture is also fairly common, providing a distinction between images which simple color-based indexing cannot provide. Configuration and shape is rarely used, because there is no clear and simple description available which can be easily indexed. The search space also increases very quickly once these dimensions are added. Systems that implement shape most often use only a simple, generalized description, such as an ellipse or a rectangle, for all shape definitions.

The alternative to using machine vision techniques is using text-based indexing. However, this requires the manual labelling of images by individuals. Because different individuals would desire different elements to be encoded, the feature of interest in a particular case may be unavailable if it was never manually encoded. Because of this, text-based search cannot be the basis of any general image search engine.

## 2.2 Image Retrieval Systems

Content-based image indexing has been proposed and implemented in several systems to date. The main image indexing features used are color, texture, and shape, with configuration of image features only rarely addressed. A good starting point on the extensive literature is the paper by Pečenović et al., [29]. The systems discussed below were looked at before or during the creation of Imagina. This is not meant to be a thorough coverage of the literature. Instead, this is a sampling of systems which have features similar to Imagina. The goal of most systems is to support queries by an example image. One system reviewed here, VisualSEEK, supports user queries via a form of “sketches”, as well as the modification of the weightings of particular features used for searching, such as color content and texture.

### 2.2.1 Query by Image Content (QBIC)

Query by Image Content is one of the earliest commercial attempts at an engine providing content-based image indexing. It was built at the IBM Almaden Research Center, and started out based primarily on color-histogram image indexing.[27] The features extracted for images are primarily global features like color histograms, global texture and the average values of the color distribution. Object features can also be used to narrow the range of retrieved images.

Images in QBIC are represented as whole images, but can also have manually outlined objects. Retrieval is performed by measuring the similarity between the user’s query and the database images. Specific similarity functions are defined for each feature. Although this is one of the least sophisticated image querying systems technologically, it nevertheless has sufficient support as a commercial system. It should be noted here that most of the other systems mentioned here, or covered in the literature, are not used commercially.

### 2.2.2 The Virage Engine of the AltaVista Picture Finder

The web-based search engine Altavista ([33]) has set up an image searching tool in collaboration with Virage, Inc.([47]) It is a very ambitious project, currently containing indexing information for over 26 million images. From those, however, less than forty thousand pictures have been indexed based on shape. The approach taken is more geared towards a text-based search based on the text presents in the web page where the thumbnail was found.

The engine does provide, however, a “visually similar” query method, which provides poor

results which seem to be based primarily on color information. A sample query is shown in Appendix C on page 129. Because a proprietary system is being used, the kind of information extracted from the image is not known for sure. It is likely that text and texture information are also used in performing the “visually similar” search.

The altavista search engine is the best representative of a production system, and is readily accessible over the internet. In a production system, the most important criterion of evaluation is the speed of matching for a given user query. For such a system, using a primarily text-based method with only basic image processing as an additional information source is crucial for providing the required performance. The Altavista Picture Finder lies at the opposite end of the spectrum from Imagina in terms of content-based image retrieval systems.

### **2.2.3 WISE: A Wavelet-based Image Search Engine**

This engine uses a wavelet encoding of the images for searching.[48] Indexing is based on a Daubechies wavelet transform which is applied to each color channel individually, as well as on color histogram matching. The search goal is to find exact image matches to an input image. The wavelet transform is applied for diagonal, vertical and horizontal directional variations. Clustering methods are then used to partition the feature space of the results.

One way to think about wavelet transforms is in terms of their similarity to the Fourier transform, a cornerstone of texture definition. Therefore, the wavelet transforms are good at determining the texture of the images they are applied to, and WISE can be thought of as a texture-based indexing system. Because of the use of color histograms, WISE also performs color-based indexing. Shape and configuration constraints, however, do not come into play outside of the textural restrictions imposed by the wavelet transforms.

### **2.2.4 Photobook**

The Photobook system was developed at the MIT Media Lab.[30][31] In its most recent version, described in [26], it includes FourEyes, a system which uses user interaction to improve indexing performance. Instead of using just one model, FourEyes instead uses a society of models, with the output being a combination of their individual results. It also uses textual labelling of regions besides image information to aid in searching.

The basis for the FourEyes extension to Photobook is that queries which go beyond color, shape

and positional cues have to incorporate a variety of features. This can be very complex, and can be dependent on a large number of variables which may not be known a priori. Therefore, a machine learning approach which adjusts performance based on user interaction over repeated usage was implemented.[26] The performance being adjusted was based on the task of classifying different image regions into different user-specified categories.

One of the central features of FourEyes is its user-friendly interface. Instead of selecting values for arcane variables, the user provides positive and negative examples from which FourEyes extracts grouping rules. In a way, FourEyes needs to be taught what to do before it can perform properly. A difficulty that arises, however, is that the number of examples for complex rules can get large. Even though the training and fine-tuning can be spread out over several usage sessions, the time investment required in order for the system to begin producing useful queries can be prohibitive from the point of view of some users.

The usage of feedback is a very promising approach to solving any challenging problem. Instead of having a powerful program a priori, a weaker program which is capable of adjusting itself to a particular usage pattern can instead be made. This is akin to biological systems, which depend on both hard-wired abilities as well as the acquisition of new skills.

### **2.2.5 VisualSEEk**

This system is one of the few which have attempted to use region configurations explicitly. This is in contrast to a system like WISE, described in subsection 2.2.3, which can only claim to use configuration information as a side effect of the image encoding used. A VisualSEEk query consists of a set of rectangular patches of color in a selectable configuration.[40] Thus only region size, not region shape, matters for matching.

This engine is designed for speed and interactive querying. Users can give feedback with regards to the which feature is most relevant to their search. The feedback is given through modifiable weights which are available for global color, color of regions, global texture, and joint color and texture matching. The system also allows for textual image retrieval, which can be used as an aid, by first doing a textual query, and then using the retrieved images as primers for retrieving further visually similar images. However, the query system is designed primarily for querying based on the location, size and color of regions.

### 2.2.6 Blobworld

The Blobworld system, described in [2], [6], [5], [9], [8] and other papers, uses an image description which attempts to step away from the global image processing approach used in most other systems. The authors (eg. [9]) recognize that most user queries are based upon the matching of objects within an image, and not on the matching of overall image qualities. Therefore, they segment each image into a set of regions, which are modeled by ellipses, and which they call blobs. The segmented regions, called blobs, have features such as color, texture, and shape. They are described in terms of their dominant features, based on an Expectation-Maximization clustering.[2]

The blobworld representation is concise, to allow rapid processing. Queries can be made either based upon an initial image, or based upon blobs selected from several images. This latter approach allows for the combination of features in the query. To improve usability, the internal image descriptions are provided to the user, giving feedback on what the query image was segmented as, and what the images retrieved from the database are encoded as. This feature is not usually found in global-property image indexing systems, because in such systems it is not straightforward how to display the encoding in a manner comprehensible to a human user. By providing feedback, the hope is that the human users adjust their queries such that the retrieved images more closely match the desired images.

## 2.3 Imagina in Comparison

Most of the features of Imagina can be found in one or more of the image retrieval systems discussed. For example, Imagina uses color-based processing, in forms of histograms and global filters. Color is a very basic feature used by all of the cited systems, except perhaps the WISE system. The contribution of this thesis on the topic of color based indexing is the provision of a thorough description of the Hue-Saturation-Value color space, in section 4.2.

The multiple levels of detail used in Imagina is another idea used in most existing systems. The success of the wavelet approach is based upon this idea. Imagina's contribution on this topic is the analogy drawn with biological processing and parallel computation, and the uniformity of the multiple level design throughout our algorithms.

A related topic, the combination of multiple methods, can also be found in previous work, usually as a byproduct of trying to use all sources of information present in an image. This is done implicitly, when an image search tool combines color and textural information. Photobook is

the only image retrieval system that explicitly discusses using multiple ‘experts,’ each of which is capable of performing image retrieval in a limited way. These experts each give an opinion as to the identity of image patches present in the image, and the consensus is used to define the identity of the image patches. Imagina pushes the idea of multiple independent methods a step further with the implementation of the shape descriptions. Instead of using a monolithic shape representation, Imagina encodes shape explicitly in four different ways that complement each other, as described in Chapter 8. Although it is absent in most image retrieval systems, shape information plays an important role in Imagina.

Blobworld is the first system that insisted on making the representation explicit. This is probably related to the fact that they are using shape-based indexing. Hence, the representation which the system uses can be interpreted by the user. We designed our system around this principle. Thus, every representation and comparison has a pixel-based debug output that can be displayed on screen.

A characteristic that is unique in Imagina is the user input in form of a freeform sketch. In the literature, a query is usually an image which is already in the database. This helps the systems run similarity measures in advance, and to simply display results from a table lookup into these previously computed results at query time. The disadvantage of such an input is the constraint it imposes on the user, limiting his versatility. In Imagina, instead, a drawing interface is provided. An image download option also allows users to bring in existing pictures from the web or drawings constructed on any other drawing tool. Together these methods allow for a greater freedom of expression in querying the system for the user.

In summary, most of the design principles that govern the construction of Imagina have already been explored to some extent in predecessor systems. Imagina is unique in combining these characteristics along with the versatility of a sketch query. The work of previous systems has helped to guide the design of Imagina into being a successful image retrieval system.

That is not to say that image retrieval is a solved problem. Most of the existing systems only address one part of a complex problem. Moreover, many basic questions like the use of color, segmentation, shape description, and recognition, remain without a robust solution. Our goal in constructing Imagina is to address these basic questions, rather than to provide a definitive solution. Along these lines, we feel our contributions to be in the areas of color representation, edge extraction and shape description.



## Chapter 3

# Theoretical Foundations

*I hear and I forget. I see and I remember. I do and I understand.*

Confucius

Current research in cognitive science has shed light upon some of the generalities and specifics of the visual processing being performed by the mammalian brain. Therefore, as a background to our approach, several topics of interest are described in this chapter, all of which have their grounding in cognitive science. The topics presented here have influenced the work in this thesis, both directly and indirectly.

### 3.1 Streams and Counter-Streams

One of the best-documented features of the visual cortex is that computation does not just cascade from the retinal sensors to the higher-level areas, but instead the information from different regions along the way is diffused bi-directionally. This double-flow of information allows higher level processing to feed back into the lower level processing. This feature has led to the proposal of a *Streams and Counter-Streams* architecture for visual processing by Ullman.[45]

Ullman hypothesizes that recognition arises in the intersection between two processing streams of opposite direction, one going ‘bottom-up’ from the retina, and the other going ‘top-down’ from memory. The input image stream is generalized at every step, allowing for more generality and

abstraction, at the same time that the stored models of images in the brain are made more and more precise, trying to match up with the current image. In the terms of Imagina, the stream is the encoding of the input images and user image queries into the indexable color and shape representations introduced in subsequent chapters. The counterstream is the set of images which are brought up for consideration from the database. By using an iterative process, some database images can be primed on one feature space, and then compared against another.

Imagina starts out with an incomplete description of the world, the same way someone waking up in the middle of the night would be encountering. The person waking up has no prediction of what is expected to be out there, so they look around, confused, for a while, looking for features. The features detected in the user input can be used to activate representations in the database, which in turn provide feedback to the user on what to draw next. This can be seen as a form of priming of the input channel.

There are many internal representations for each object being identified by the human visual system. This can be deduced from the variety of neurological impairments brain damage can produce.[35] Thus, in general, there is no single mechanism which will succeed at providing general matching. Instead, every mechanism available will succeed in some cases and fail in others. If the mechanisms applied fail independently, then a more robust system can be achieved through the combination of a set of more failure-prone modules. This is the approach taken in Imagina.

## 3.2 Gestalt Psychology

The Gestalt psychologists categorized different features which humans can rapidly pick out in visual input. Their approach, however, was lacking in a model for why their observations held. Instead of looking at the human visual processing as the end of a process, they looked at it as a whole. Thus, although they described patterns of processing, such as the grouping of visual stimuli, or the segmentation of line drawings based on ‘good continuation,’ they never attempted to go beyond these descriptions of their observations.

However, this knowledge can nevertheless be applied fruitfully to an image indexing project. Even without the understanding of the processing performed by the brain, algorithms which compute similar outputs can be devised. A simple application is the use of edge filters to redescribe an image in terms of the output of the filters. This can be done by overlapping the edges which have been detected and drawing them onto the new image. The new image would then contain a

re-description of the original in terms of edges. The use of overlapping would fill up gaps in lines, thus allowing the system to detect imaginary contours which humans would see.[49]

Approaches using information derived from or similar to the observations provided by Gestalt psychologists have also been developed previously. For example, some such patterns are described by Ullman in [45] as “salient features.” He argues that primitive routines can be applied to the image to extract the different pieces of information that can be used in matching images. These different pieces of information combined with an abstracted version of the image constitute the matching criteria for a query.

### 3.3 Canonical Views

In a series of studies, Bülthoff and others have shown that objects are most likely stored as multiple canonical views in the human brain.[4] They theorize that these canonical views are then interpolated between to allow the recognition of a wide variety of views. Support for this theory can also be found in [45], where a method for interpolating between sketches of cars is presented. There, Ullman describes how views taken of a car at thirty degree angles can be interpolated between linearly to predict with minimal error any view in between. Other work, by Ullman and others, has shown this method to be extensible and reliable, as for example in [36].

The central theme in this body of work is the use of prototypes, both for individual object classes, as well as for individual objects. The same theme can be found in the cognitive science literature, for example in the classification work of Eleanor Rosch.[46] For Imagina, this concept can be applied in terms of the database structuring. For small databases, all images can be indexed individually, and compared to all other images. As the database grows larger, however, methods for reducing the computation required for a new image to be inserted or indexed need to be introduced. Clustering images and image regions into a hierarchy of prototypes would allow for quick indexing, similar to the binary space partition trees used in vision algorithms.

The line of experimentation also provides evidence for differences in encoding between rotations around the horizontal and vertical axes, with the latter being recognized more quickly. These results can be useful for an image indexing problem in allowing a greater distortion along the horizontal axis, equivalent to rotations around the vertical axis, more so than distortions along the vertical axis, equivalent to rotations around the horizontal axis. This would match the user’s tendencies to be able to compensate for deformations in the originally viewed image. Since the user is expected

to be able to compensate more for deformations along the horizontal axis, their memory for this axis is likely to be less accurate to the exact pixel-level image which is retrieved.

### 3.4 The Limitations of Passive Input

In his paper on active touch ([10]), Gibson argues that the experience of being touched is fundamentally different from the experience of touching in its ability to provide information. Active touch is defined as an exploratory rather than merely receptive sense. It is more like scanning, using touch, than just perceiving. Because the change of internal state caused by muscle and limb movement is correlated to a change in the input, a more thorough description of the input is available. The feedback received by the limb being used as a probe is more informative to a human than simply rubbing the same object over the same area while the limb is held passively.

From this observation, one can argue that active probing is required for the ability to correlate internal and external states. In turn, this is a requirement for learning, and for the ability to understand. Therefore, a system which can only passively accept input is at a loss when trying to gain an understanding of the input beyond a simple stream of excitation. But this is exactly the case that an image indexing system finds itself in. If the system only has a set repertoire of actions, such as segmenting input images in a particular deterministic fashion, then it can never acquire any new information. It can only be as good as it was when first created.

This argument can then be followed through in two ways. One way is to argue that because an image indexing system cannot *look* around, and therefore cannot change its input, it cannot ever learn to abstract from the images beyond the pixel values. Therefore, the best one can achieve in a static system is some approximate accuracy based on image correlation measures which are valid for the task at hand. For example, if the task is to find the waterfalls in the image, then a deformable template which matches narrow regions of blue with white near the bottom, surrounded by greens or browns on the sides, would be the ideal measure. In fact, this was done as part of a PhD thesis, as described in [25]. Alternately, if the task is to identify the nut in a precision machine-tooling environment, pattern matching with rotation is likely to be the best solution. This is commonly done in robot vision applications for factory settings.[13]

On the other hand, an image indexing system may be able to get beyond its envelope. Although it is stuck in a 2-dimensional world, because it cannot ever probe beyond each distinct image, it does have another input which it can take advantage of. This is the human user. This is akin to a person

who is in the dark using their hearing or touch to make their way to a target. Through the human's actions, which matches are good and which are bad can be learned. This idea is only beginning to be applied to content-based image indexing, for example in the Photobook implementation.[26]

### 3.5 Layering of Processing

One central idea which is found in biological processing is the use of several simultaneous levels of processing being applied. This can be found, for example, in vision, where different cells respond to bars of different widths in the visual input. At higher levels of vision, these response functions are somehow combined to provide size invariant object recognition.

Because the different computations occur in parallel, features can be detected regardless of their actual sizes at any one time. A specific feature size does not have to be pre-selected as a desired input value. Instead, the presence of a feature in the input can be deduced based on the responses of the size-specific feature detectors.

The computer science application of the concept of having layers of hierarchical representation can be found in several existing applications, for example in the computation of optical flow in computer vision. In this application the optical flow is computed on a coarse scale version of the image first. The computed optical flow is then propagated to the next larger scale version of the image, and used as a guide for the computation of the optical flow at this level. Eventually, after having passed through all of the coarser versions of the image, in what is termed a pyramid, the propagation reaches the image itself. With this progressive approximation technique, the computation is usually more robust and reliable.

The abstraction which can be taken away from such applications is that a cheap approximation computed on a coarse scale version of an input can be used to improve the desired computation on the fine scale input. This approach can also be extended to a parallel architecture, where instead of a cascade an interaction between levels is used, based on excitation or inhibition between the different levels.

### 3.6 Combining Unrelated Measures

The idea of how to combine unrelated measures comes from a reading of the paper by Itti, Koch and Niebur, [15]. In it, they discuss how saliency can be computed based upon the combination

of several measures, each of which are computed across an entire input image. Their solution to combining the different measures is that measurement spaces which have only a few peaks across a single image are weighted more heavily than measurement spaces which contain either a lot of peaks or no peaks. This paper is discussed further in section 6.4.1 starting on page 64.

How to combine a set of different measures is always a difficult problem. This issue crops up in any system which attempts to use a combination of multiple unrelated computational methods to solve the same problem. There is no straightforward solution, other than having weights which are either pre-set based on a training set of data, or which are modifiable by the user at runtime. In a system like Imagina, this is a difficulty which must be addressed.

A very powerful idea which can be applied to this problem is that the different measurements which are to be combined should be weighted based upon their respective measurement spaces. Basically, the deviation of each measure from its expected value is the most reliable weighting for it. Because each measurement space has a different distribution, this distance should be normalized based on the standard deviation of the measurement space. It is helpful if the shape of the distribution of values is close to the normal distribution. However, a more complex procedure could be applied in other cases.

In terms of signal analysis, the different measurements have to be compared against the background noise in their respective measurement spaces. A value which is further from the mean is more likely to be the result of a signal, as opposed to the noise which is in the background. The larger the value measured, the more likely it is to be a part of the signal being sought. Therefore, the further the value is from the mean, the more reliable that value is as a measurement, and the higher its weighting becomes.

# Chapter 4

## Color

*People only see what they are prepared to see.*

Ralph Waldo Emerson

Images are made up of a set of colored points which are positioned on a rectangular grid. To extract any information from this grid, a good understanding of what the different values of the colored points represent is required. This is especially important since this is the basis of all subsequent processing in any image retrieval system, including Imagina.

This chapter first addresses the issue of color by discussing the two color spaces which are most commonly used in image retrieval systems, the Red-Green-Blue (RGB) and Hue-Saturation-Value (HSV) color spaces. The advantages and disadvantages of these colors, as well as methods of using the color information without any pre-processing are then discussed.

### 4.1 The Red-Green-Blue (RGB) Color Space

The predominant color representation used in Imagina is the RGB color space representation. In this representation, the values of the red, green, and blue color channels are stored separately. They can range from 0 to 255, with 0 being not present, and 255 being maximal. A fourth channel, alpha, also provides a measure of transparency for the pixel. Although useful for displaying overlapping

image graphics and visualizing algorithms during testing, this channel is not used by any final image processing algorithms in Imagina.

The distance between two pixels is measured as:

$$\frac{(255 - |\Delta Red|) \cdot (255 - |\Delta Green|) \cdot (255 - |\Delta Blue|)}{255^3} \quad (4.1)$$

where  $\Delta Red$  is the red channel difference,  $\Delta Green$  is the green channel difference, and  $\Delta Blue$  is the blue channel difference between the two pixels being compared. This distance metric has an output in the range of [0-1]. When used to compare pixels, a cutoff, usually between 0.6 and 0.8, is used to decide whether the two pixels are similar to each other. Because the distance measure is based on the absolute difference between pixel color values, the size of the neighborhood of similarity for a given cutoff value is the same regardless of the location of the pixel value being compared to in the RGB color space.

Although this color space description does not always match the intuition for color similarity of people, it does provide an easy method for pixel value comparison. In most cases it performs agreeably well, as it can differentiate gross color differences. It is also computationally cheap as compared to any other color space representations, because the Red-Green-Blue color channel differentiation is used in most image formats and therefore is readily available.

## 4.2 The Hue-Saturation-Value (HSV) Color Space

The alternative to the RGB color space is the Hue-Saturation-Value (HSV) color space. Instead of looking at each value of red, green and blue individually, a metric is defined which creates a different continuum of colors, in terms of the different hues each color possesses. The hues are then differentiated based on the amount of saturation they have, that is, in terms of how little white they have mixed in, as well as on the magnitude, or value, of the hue. In the value range, large numbers denote bright colorations, and low numbers denote dim colorations.

This space is usually depicted as a cylinder, with hue denoting the location along the circumference of the cylinder, value denoting depth within the cylinder along the central axis, and saturation denoting the distance from the central axis to the outer shell of the cylinder. This is depicted in Figure 4-2 on the left. This description of the HSV color space can also be found in [40].

This description, however, fails to agree with the common observation that very dark colors



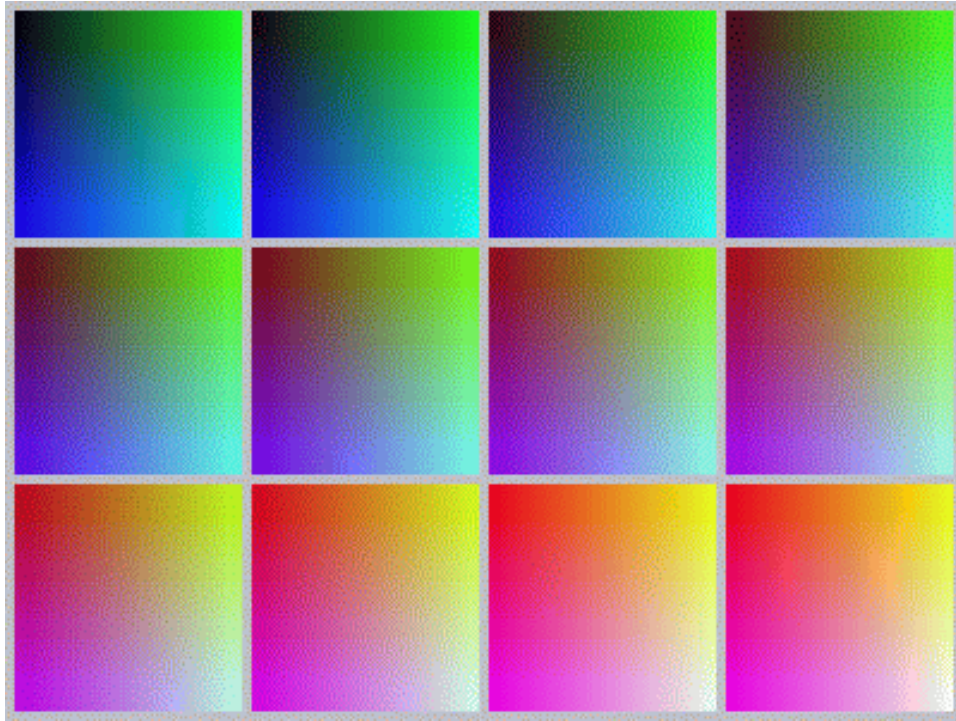


Figure 4-1: The RGB color space can be thought of as a cube, with each axis of the space representing a color intensity, from 0 to 255. The slices displayed are taken along the constant-red plane, from low to high. In each plane shown blue increases to the right, and green downwards.

are hard to distinguish, whereas very bright colors are easily distinguished into multiple hues. A system more accurate in representing this is shown in Figure 4-2 on the right, which is the model used for the HSV color space representation in the Imagina system. Similar systems have also been arrived at by others, for example [5].

In the latter representation, dark colors, which have a low value, are all considered to be very similar. On the other hand, brighter colors, which have a high value, are more easily distinguished. Although this system is closer to the observed human color space, it is not an exact model. Its simplicity makes it computationally tractable, while its better abstraction of the color space provides an improvement in performance.

#### 4.2.1 Computing the HSV Representation

The starting representation is in the RGB color space format, since this is the format that most images are stored in. The computation of hue, saturation, and value can be performed as detailed

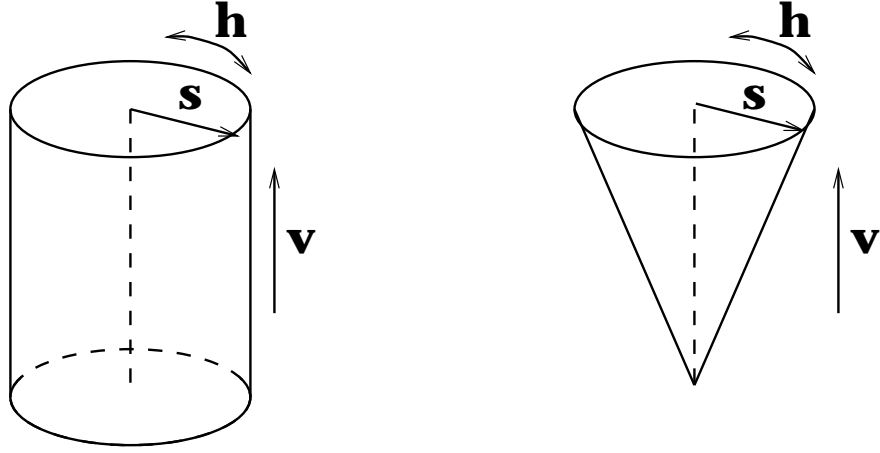


Figure 4-2: Two HSV color space representations. The cylindrical representation is a good approximation, but the conical representation results in better performance.

below. This computation, with corrections, was borrowed from[40]. In the text below, as in the figures,  $h$  stands for hue,  $v$  stands for value, and  $s$  stands for saturation. The RGB colors are similarly represented, with  $r$  for red,  $g$  for green, and  $b$  for blue. The remainder of the variables are temporary computation results. All of these computations have to be computed on a pixel-by-pixel basis.

$$v = \max(r, g, b) \quad (4.2)$$

$$s = \frac{v - \min(r, g, b)}{v} \quad (4.3)$$

Here,  $\max$  denotes a function returning the maximum value among its arguments, and where  $\min$  denotes a function returning the minimum value among its arguments. The hue is the most complex to compute, as it involves a mapping from three dimensions to a single continuous linear dimension, which can be thought of as wrapping around the perimeter of the color space cone.

$$6 \cdot h = \begin{cases} \text{if } r = \max(r, g, b) \text{ and } g = \min(r, g, b) \text{ then } 5 + bpoint \\ \text{if } r = \max(r, g, b) \text{ and } b = \min(r, g, b) \text{ then } 1 + gpoint \\ \text{if } g = \max(r, g, b) \text{ and } b = \min(r, g, b) \text{ then } 1 + rpoint \\ \text{if } g = \max(r, g, b) \text{ and } r = \min(r, g, b) \text{ then } 3 - bpoint \\ \text{if } b = \max(r, g, b) \text{ and } r = \min(r, g, b) \text{ then } 3 + gpoint \\ \text{if } b = \max(r, g, b) \text{ and } g = \min(r, g, b) \text{ then } 5 - rpoint \end{cases} \quad (4.4)$$

where

$$rpoint = \frac{v - r}{v - \min(r, g, b)}, \quad (4.5)$$

$$gpoint = \frac{v - g}{v - \min(r, g, b)}, \quad (4.6)$$

and

$$bpoint = \frac{v - b}{v - \min(r, g, b)}. \quad (4.7)$$

The value computation, as defined in equation 4.2, maps to the range of values which  $r$ ,  $g$ , and  $b$  can take. This can be normalized to be in the range of [0-1], if desired. The saturation computation automatically maps to the range of [0-1], with the border cases occurring when the minimal value is 0, and when the minimal value is equal in magnitude to the maximal value among the RGB values, respectively. Therefore, the hue computation, as defined in equation 4.4, maps every RGB value to a different location in the color cylinder defined in Figure 4-2. To map into the cone representation, the saturation  $s$  has to be multiplied by the value  $v$ . In Imagina, this computation is done at the time of matching, instead of at the time of representation building. Note that the equation given computes  $6 \cdot h$ , not  $h$  directly. Thus the range of hue is [0-1], with 0 being equivalent to 1, resulting in a circular value space. The HSV transformation is fully invertible, so no information is lost through this transformation.

#### 4.2.2 The HSV Distance Metric

The distance between two colors can be computed in several ways. The most obvious approach, especially when using the cylindrical HSV color space description, is to use raw distance. This

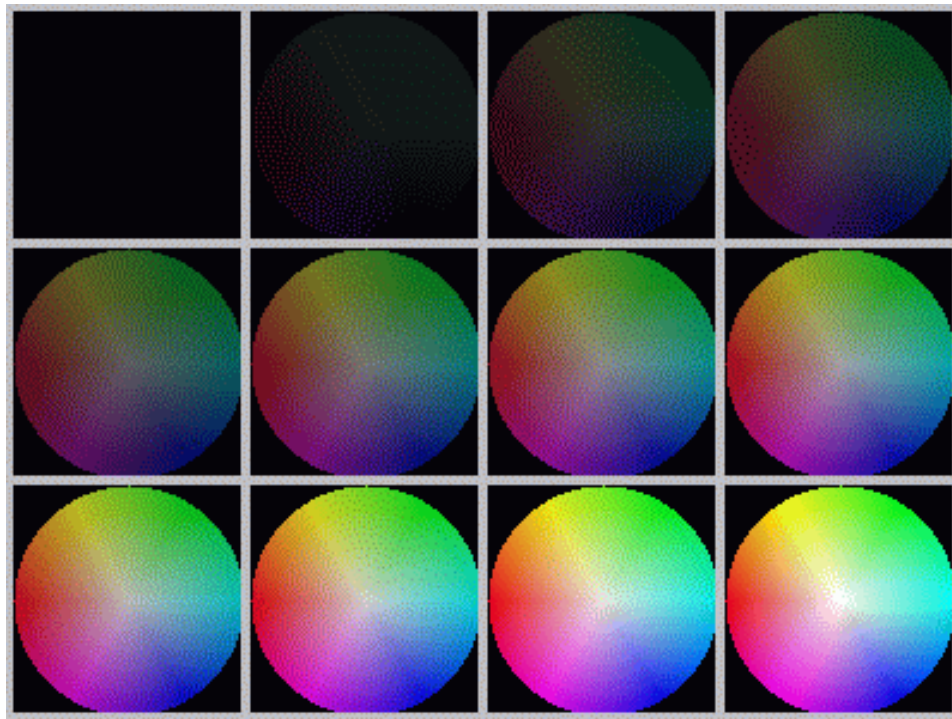


Figure 4-3: The slices displayed are taken for single-value planes through the HSV cylindrical color space representation. The hue is the angular position of each point with respect to the center of each slice. The saturation is the distance from the center of each slice.

results in a distance metric which considers colors which are located in a sphere around a point equally similar to that point. Because the axes of the HSV color space are not orthogonal, this description is not satisfactory, however.

Although many variations on this distance measure are possible, let us first consider what properties are desirable of such a measure. The ideal distance measure would have several features:

- Very dark colors, which have a low value, should be considered very similar.
- In turn, bright colors, which have a high value, should be more differentiable.
- Colors with very low saturation should be considered to be similar regardless of hue.
- The distance range should be upper- and lower-bounded.

The measure settled upon for Imagina, which has an output in the range of [0-1], with 1 being most similar, and 0 being completely dissimilar, is

$$\sqrt{\frac{1}{2} \cdot (s_1^2 - s_1 s_2 \cos(\theta) + s_2^2 + (\Delta v)^2)} \quad (4.8)$$

where  $\Delta v$  is the difference in value, and  $s_1$  and  $s_2$  are the saturations in the conical, not cylindrical, HSV color space. The transformation is defined as  $s_i \cdot v_i$ , for the respective similarity of the two pixels  $i = 1, 2$  being compared to each other. The weighting of the saturation results in darker colors being considered to be more similar than lighter colors, which is a known perceptual phenomenon. The multiplication by  $\frac{1}{2}$  is used only to map the distance values to the range of [0-1], and is determined by dividing 1 by the largest distance possible. In the equation, both saturation and value range from 0 to 1. Therefore, the largest distance possible is the diagonal of the circle of maximal saturation at the top of the cone, which is 2.

$$\theta = 2\pi \Delta h \sqrt{\max(s_1, s_2)} \quad (4.9)$$

The computation of the  $\Delta h$  is not straightforward, because of the circularity of the  $h$  value space. Given two pixels A and B,  $\Delta h$  is computed as:

$$\Delta h = 2 \cdot \begin{cases} \text{if} & |A.h - B.h| > 1/2 \quad \text{then} \quad 1 - |A.h - B.h| \\ \text{else} & |A.h - B.h| \end{cases} \quad (4.10)$$

Since the hue space is circular, the absolute distance between two hues is at most half the circumference of the hue space. The multiplication of  $\Delta h$  by  $\sqrt{\max(s_1, s_2)}$  in equation 4.9 decreases the magnitude of the angle between the two colors being compared, and therefore increases their similarity, in direct relation to the saturation of the two pixels. The square root is then taken so that the saturation weighting rises to 1 faster than linearly. As a result of the weighting, colors which are saturated very little, and are close to the white-gray axis, are always similar regardless of hue. Colors which are very saturated, and which are very distinct from the colors in the range of white to gray, on the other hand, are always distinct. Because the max function is used, the comparison between two pixels is the same both ways.

By weighting the saturation values used throughout this computation, a true distance in conical space is created. The only difference between a pure euclidean distance and the distance being used is that the hue angle gets modified by the saturation of the pixels being compared. This results in the elongation of the matching space along the hue but not the saturation or the value axes of the space. This property is desirable due to the human visual clustering preferences.

Two new variables, *SaturationWeight* and *ValueWeight*, can be introduced in order to weight the distance measure being used, and in turn to stretch the space in a particular direction. With the definition given above, and used in Imagina, the value dimension has a height of 1, while the diameter of the saturation dimension at the top of the cone is 2. To change this, every  $s$  in equation 4.8 can be multiplied by *SaturationWeight*, and the  $\Delta v$  can be multiplied by *ValueWeight*. By using these two variables, the importance of saturation and value weightings can be changed. The fractional multiplier used to achieve a [0-1] distance range,  $\frac{1}{2}$  in Equation 4.9, however, would need to be modified accordingly, to be 1 divided by the maximum distance possible in the stretched space.

### 4.3 Other Color Spaces

The RGB and HSV color space descriptions are not the only ones applied in the literature. For example, the WISE system described in [48] uses a color space description where the RGB color space is converted to a new 3-color color space description:

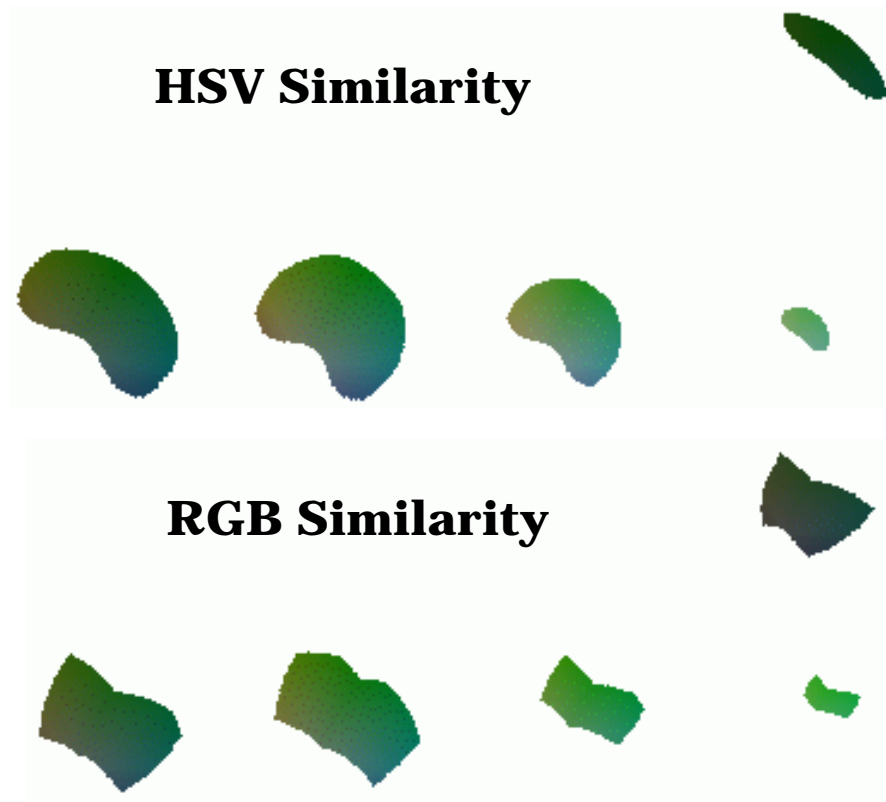


Figure 4-4: A comparison of the similarity measures of the RGB and HSV color spaces. The colored points shown are all of the points considered similar for a given similarity cutoff value. The cutoff value has to be different in the two spaces in order to result in approximately equal areas of coverage since the two measures are unrelated. The actual cutoffs used are 0.88 for HSV and 0.74 for RGB. The colored points were selected from the HSV display in Figure 4-3, in comparison with a point roughly at the center of the colored points selected in by either similarity measure.

$$\begin{aligned}
C_1 &= \frac{1}{3} \cdot R + G + B \\
C_2 &= \frac{1}{2} \cdot R + \max(R, G, B) \\
C_3 &= \frac{1}{4} \cdot R + 2 \cdot (\max(R, G, B) + B)
\end{aligned} \tag{4.11}$$

A more thorough discussion of color spaces can be found in [40]. Because the HSV color space is the simplest description which is similar to the color space of humans, these other color spaces were not explored beyond their description in the literature.

## 4.4 Use of Color Spaces in Imagina

The algorithms written and tested use primarily the RGB color space representation. Although the HSV color space representation is more accurate in terms of human perception, the efficiency of the RGB color space computations far outweigh the improvement achieved through the use of the HSV color space computation on images. In particular, both the initial HSV color space mapping, as well as the subsequent distance metric calculations, are very computationally intensive. Further, some comparisons are more easily computed in the RGB color space, requiring repeated conversions of the color space representation.

The RGB color space is used in the earlier color-based segmentation algorithm implementations. It is also used whenever color averaging over a region is needed. Although it is not clear whether this is equivalent to averaging in the HSV color space, the RGB color space averaging is much more easily computable. The HSV color space is used in the algorithms which are in the final Imagina implementation, especially where the comparison of point values is required.

## 4.5 Color-Based Indexing

Color is the primary method of indexing used in most content-based image search engines, because it can be described compactly and performs very well for the limited amount of data storage required. It is also the most straightforward in conceptual and implementation terms. However, it is very hard to get color-based indexing to work right in terms of human visual judgments.

Color-based indexing can be done both for whole images as well as for subsections of images. Even regions extracted based on color can have different spectral distributions within the color range allotted to them during segmentation. The utility of color-based indexing is limited, however, by



the accuracy of the color distribution in the input image as compared to the color distribution in the desired image. For user-drawn sketches, the color distribution may be a very poor match to the desired image. Therefore, this measure is most useful for queries based on real images.

Two methods of color-based indexing are used in Imagina. The first is based on color histograms, where a set of bins are defined over a given color space, and then similarity is based upon the distance between images in the redefined space. The second is based on a color map, where different scale versions of the two images being compared are mapped to each other on a pixel level, yielding a comparison of the color distribution within the region itself. These two methods are complementary, in that where one method fails, the other succeeds.

Color can also be used for the determination of coherent regions within an input image. This application of color is explored further in Chapter 6, where the process of segmenting an image into regions is described.

## **4.6 Comparison Based on Color Histograms**

Color histogram use for image indexing was re-popularized in the early 1990s by their use for real-time image matching.[41][42] However, this system works well only if a large number of pixels is available, or if an accurate color model is used. For the implementation used in Imagina, a binning technique similar to the implementation QBIC uses was implemented.[27]

### **4.6.1 Discussion of Previous Approaches**

Several ways of computing color histogram matches exist. A thorough treatment of the different methods of computing matches is found in [40]. The method which was settled upon is the cosine distance measure, because this is a measure which is size-invariant, and which usually performs well in vector computations. This is true, for example, in text-based processing, where different pieces of writing can be most successfully matched using the cosine distance between multi-dimensional vectors denoting the words used in the writings.

The difficulty with the comparison of histograms is that the bins have to define ranges of color which are perceptually distinct and very similar to each other. Thus, colors which are perceptually similar should be binned together. To deal with this paradox, and to improve performance, QBIC and VisualSEEk define a similarity matrix between the bins. Then, when two histograms are compared, every bin is compared to every other bin, and their equivalence is weighed by the given

amount. The sum of these comparisons is then returned as the equivalence value. This approach, unfortunately, requires that individual bins be compared and that the ideal weights be evaluated.

A simplification commonly applied to the color histogram approach is the use of color sets. A color set is created by thresholding the color histogram values, assigning ‘1’ to the respective color set if the histogram entry is above the threshold, and ‘0’ otherwise. These vectors can then be compared more efficiently than the color histograms. Further, this reduces the variations caused by underrepresented colors in a given region. Because speed is superseded by functionality in terms of the goals adhered to in Imagina, this simplification was not implemented.

#### 4.6.2 Color Histogram Definition in Imagina

Imagina uses an implementation which attempts to address these issues. Unlike systems such as QBIC and VisualSEEk, which use the cylindrical HSV color space, Imagina uses the conical HSV color space. Because of this, the definition of bin boundaries is not as straightforward. However, the resulting bins separate colors into more distinct sets, which makes matching more efficient.

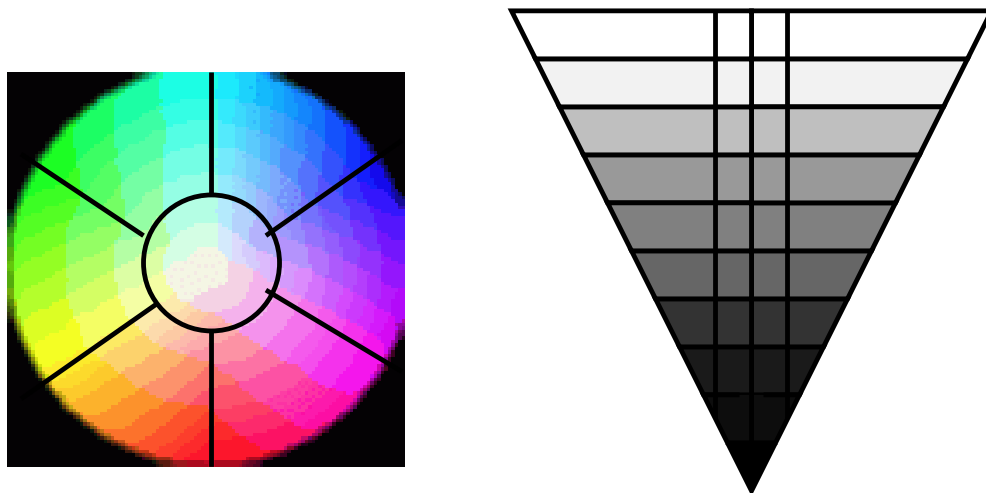


Figure 4-5: Two slices through the HSV color space, showing two views of the binning. The left slice cuts the cone through a single-value plane. The right slice cuts the cone in half through the central axis. The left slice denotes what is referred to as ‘horizontal’ partitions in the text, and the right slice denotes what is referred to as ‘vertical’ partitions in the text.

The HSV color space is split up into several layers along the value axis, with each layer split up into an equal number of bins. These are split based on a perceptual distinction into two sets. The first set of bins is defined around the central axis of the HSV color space, where the saturation is

close to zero and perceptually the color range from black to white is present. These bins are referred to in this text as ‘central bins.’ A second set of bins is defined in a circle around the first set of bins, with each bin at a given level containing a range of the hue range at that level.

The bottom-most level, representing colors very similar to black, is made up of a single central bin. The number of partitions both vertically and in the plane of a single value range can be parameterized. The number of bins used in Imagina, depicted in Figure 4-5, is 64, with six outer partitions and ten vertical partitions. This results in nine levels with seven bins each, plus the bottom-most level’s single bin.

Figure 4-6 shows the format of the color histogram display. In the display, each bin is represented as a vertical slice. The bins are arranged by hue similarity, with all of the bins which are in the same vertical stack being displayed adjacent to each other, going from darkest to lightest.

For each bin, the color of the vertical lines displayed is the average color in the bin. The size of the bar from the horizontal lines displays the other four measures, with the length of the bar from the horizontal lines denoting the magnitude of the value of the current bin as compared to the magnitude of that value the maximal bin. In other words, this is computed as:

$$\text{barlength} = \frac{\text{currentvalue}}{\text{maximalvalue}} \quad (4.12)$$

The display shows, in order, the number of points stored, the standard deviation of the hue, the standard deviation of the saturation, and the standard deviation of the value of the points which fell into the given bin. Because the non-central bins become larger at larger vertical positions, the standard deviation of value for a uniform representation of the HSV color space would be expected to be larger for the bins representing larger values, which are to the right, than the bins representing smaller values, which are to the left, within each bin set displayed. For hues, the deviation should be the same within a given bin set, because they always contain the same hue range. This histogram pattern, and a comparison with the RGB color space pattern, can be seen in Figure 4-7.

### 4.6.3 Color Histogram Matching

The straightforward method of simply computing the dot product between the color histogram definitions does not work very well because the information of the relative positions of the different histograms is lost. Other systems have addressed this problem in different ways. The primary approach is to use a matrix of weights which defines a similarity between every pair of histogram

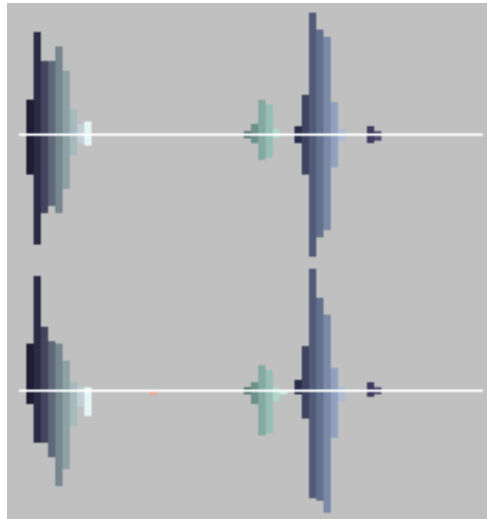


Figure 4-6: A sample image and its color histogram. The information stored in the color histograms is displayed in terms of five variables for each bin: the average color, the number of points stored, and the standard deviations of the hue, saturation and value of the points.

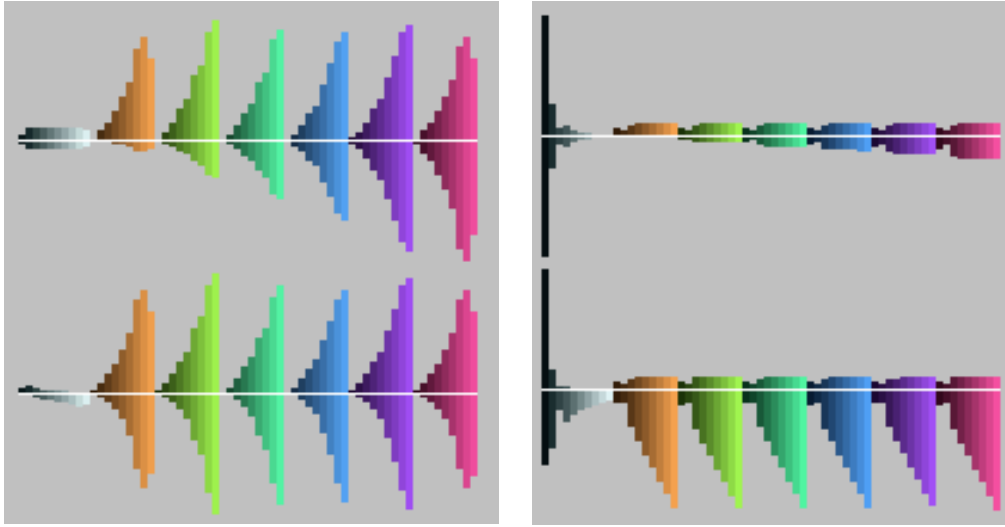


Figure 4-7: The color histograms of the displays of the RGB and HSV color spaces, shown on the left and on the right, respectively. The slight variation in the distributions of the different vertical bin sets in the RGB display is caused by the sampling error introduced by the RGB color space display. The HSV space histogram shows that the displayed values are unnormalized hue and saturations. Thus, except for value, the variation shown takes place primarily in the central black-to-white bin set.

bins. This matrix has to be pre-determined based on psychophysical or experimental data, or can be approximated as the similarity between every pair of bins based on a given measuring method.

An alternate approach is to take advantage of what is already known about the color space being used. Perceptually, bins which contain points in the same hue range, but which have different value ranges, look very similar to human viewers. The only noticeable distinction, over a large range of values, is that one range is darker than the other. This is perceptually a smaller difference than a change in color.

Bins which have different hues but the same value ranges, however, are very different. Therefore, matching can allow different value ranges of a given hue range to interact, while different hue ranges are kept separate. This is achieved by grouping together all of the bins of a given hue for matching. The bins representing the black-to-white range also make up a distinct group.

In performing this computation it is assumed that the distribution of color is more relevant than the actual lightness of the colors. Using this approach, a lighter and a darker image of the same scene or object will be considered more similar than if this assumption were not made. As an extension, hue ranges which are adjacent may also be allowed to interact, as an improvement to

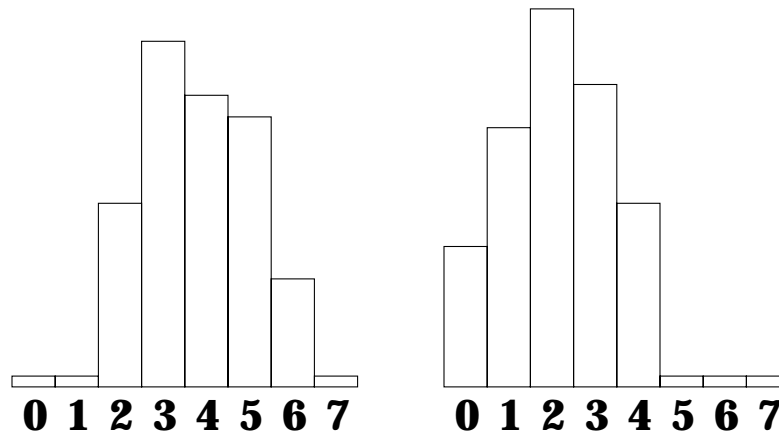


Figure 4-8: A sample color representation of two different regions, showing only the bins in a single hue range. Although the two color distributions are about the same, the shift in the bins represents a darker image on the right. In this figure the bin numbering goes from darkest on the left to lightest on the right.

this approach. This was not explored in depth as the scheme as described here works sufficiently well for color matching.

The advantage of this approach is also its weakness. Sometimes the distinction between lighter and darker views is important, as for example when comparing a white and a black background. However, the next color representation described in this chapter, based on color mappings, does not have this difficulty. Since both are applied when matching is performed, the potential misjudgment of one color representation are mitigated by the judgment of the other color representation.

## 4.7 Comparison Based on Color Mappings

A color mapping is a very simple and straightforward representation based on color. The intention of the color mapping representation is to measure not only the color ranges represented in an image, but also the their distribution. This is in contrast to the former representation, where the location of the color values did not matter.

The representation is built by taking the input image or region and resizing it into a square. Images are always subsampled to fit into a square whose sides are a power of two. This square is then iteratively subsampled into a square whose sides are half of the previous square's sides. The subsampling performed is equivalent to a RGB color space averaging filter. The iterative halving of the image stops at a square with a side length of two.

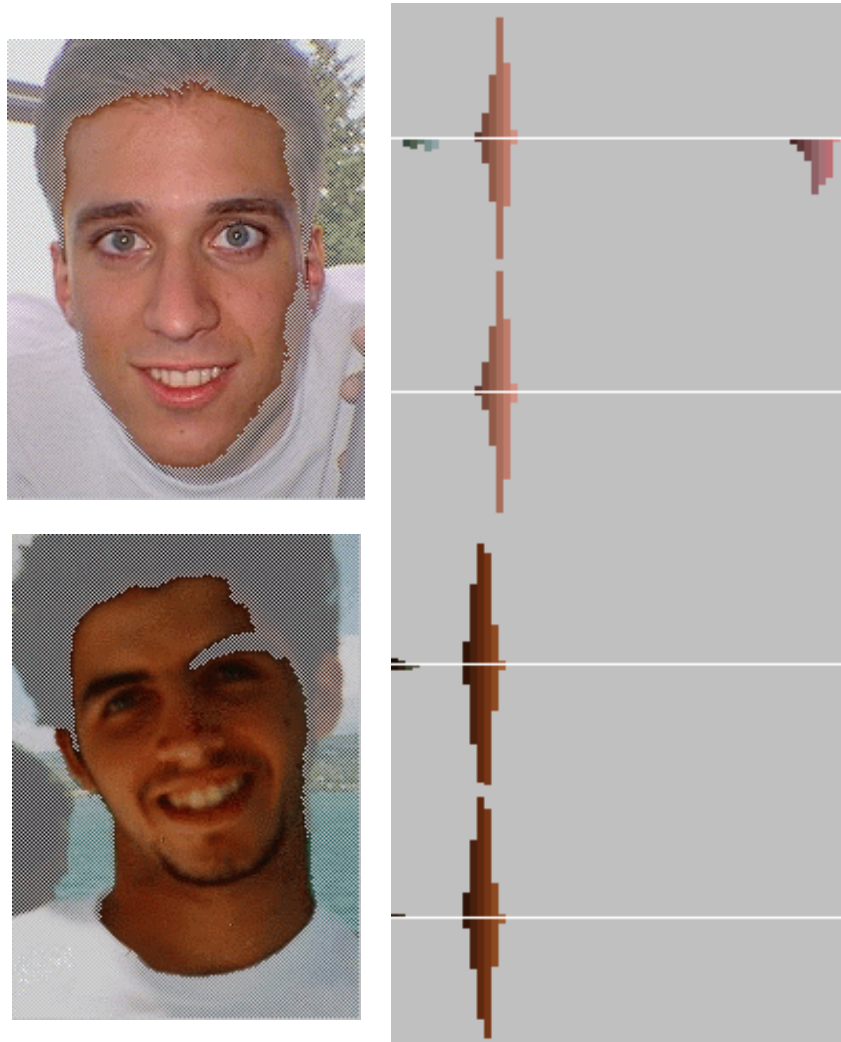


Figure 4-9: An actual color histogram match. Although the two faces are very different in lightness, both of them have a similarly shaped distribution of color in the pink hue color range, resulting in a high match value, of 0.97. This is in contrast to the figure following.

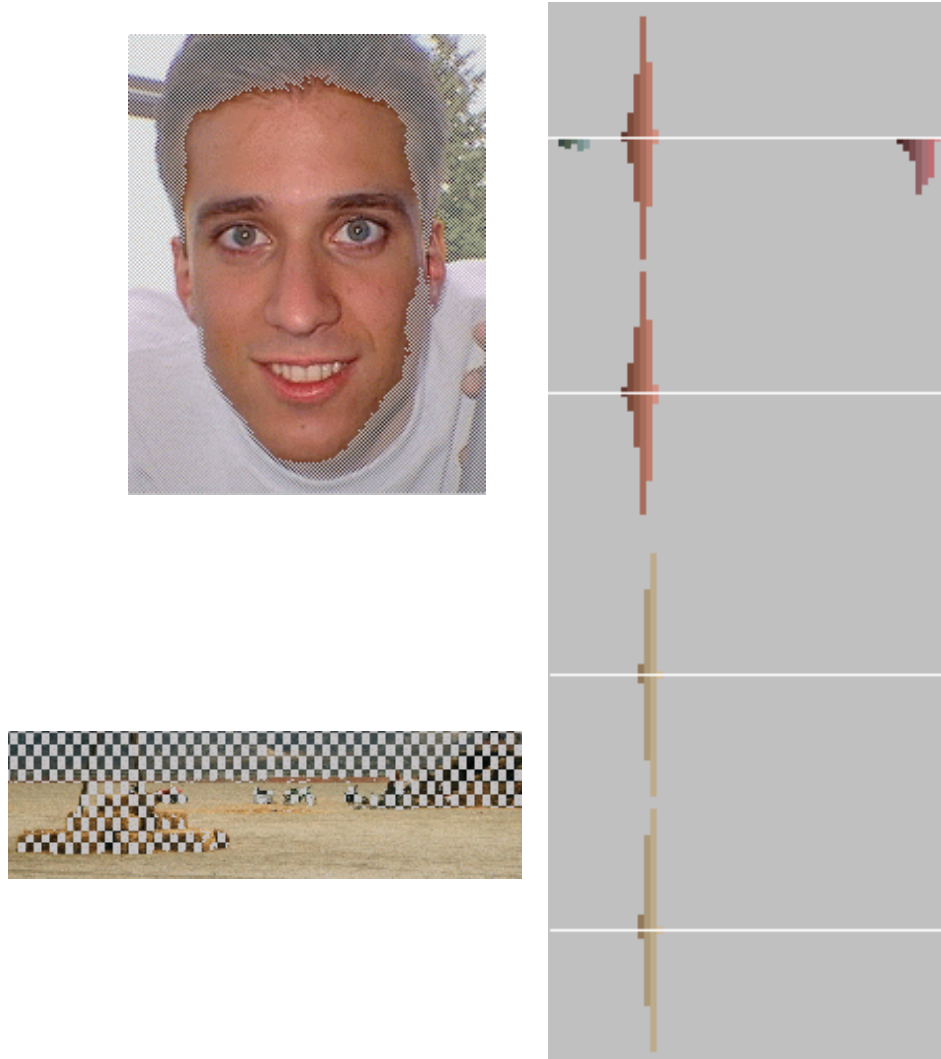


Figure 4-10: The comparison of one of the faces with a third region, which also has colors primarily in the same bin range, but whose distribution is different. The match result, 0.86, is lower because of the difference in the distribution shapes. A comparison with a region which has little of the pink hue color range would result in a value close to 0.





Figure 4-11: An image and its color mapping. The color mapping representation takes its input and downsamples it into a sequence of squares. Matching between images can then be performed by comparing the square representations of the two images being compared.

The color mapping is a representation which does not account for the actual shape of the regions being matched. All of the inputs are stretched into the same shape. Matching is then performed from the smallest level upwards, with the weighting of the match at each level being a weighted sum of the match at the previous level, and the match at the current level. The match at a single level is computed to be the average of the similarity measures of the overlapping pixels in the two color mappings being compared.

Although this representation might seem inefficient, an efficient implementation can be achieved in one of two ways. Either the entire representation is built as described above, and matching is performed by averaging a random sample of the pixels in the two representations, or the maximum square side size is limited to 16. Even at very low resolutions, on the order of 256 pixels per map, the matches between the single levels of the color mappings are within 10-15% of the matches computed when comparing the entire color mapping levels. Thus, only a small amount of accuracy is lost when limiting the matching to a small color mapping size. Although the computational efficiency is not a key issue in the design of the current system, for a larger production level system this becomes important.

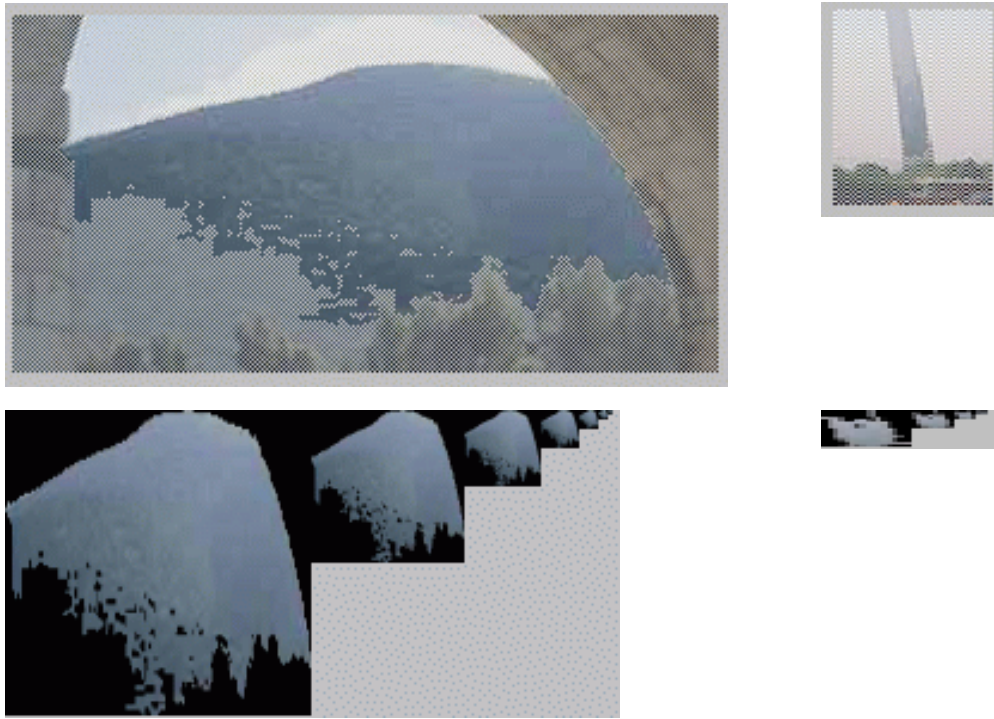


Figure 4-12: A color mapping comparison. Even though the shapes of the two regions are very different, and the original regions started out at different scales, the overall match is considered to be very good because the colors of the two regions match so well. Numerically, the match is 0.825.

## 4.8 Uses of Color-Based Comparisons

Color-based comparison is most useful when the input is most true to the color of the desired image. For drawn user queries, this is rarely the case. Although color can be specified, it is often more distant from the desired color than a real image input would be. It is also usually very evenly distributed, with each region having only one shade. Thus, the color comparison based on the distribution shape would not be very helpful in this case.

The only aid that the color comparison can provide is to distance the space of really bad matches from the space of good matches. By selecting a color, the user emphasizes all of the regions in the database which have a similar color, while regions of very different colors are de-emphasized. Therefore, color can be used as an aid in pruning the search space. However, color cannot be depended upon to perform the search when a user query is used in the stead of an actual image.

If a real image is used as a query, then color is clearly one of the best ways to perform matching. The color distribution would then matter much more.

The central utility of color, however, is in its aid to further processing of the image. Color is the basis of filtering, which is described next, and segmentation, which is following. All of the higher level representations of image regions which are developed in Imagina depend upon a good way of separating colors in an image.

## Chapter 5

# Filters

*Hither the heroes and nymphs resort,  
To taste awhile the pleasures of a court;  
In various talk th'instuctive hours they past,  
Who gave the ball, or paid the visit last;  
One speaks the glory of the British Queen,  
And one describes a charming Indian screen;  
A third interprets motions, looks and eyes;  
At every word a reputation dies.*

Alexander Pope, The Rape of the Lock

The central problem with retrieving information out of an image is that it is very hard to distinguish relevant information from noise. This is especially true for extracting meaningful and coherent regions from an image, which is the central problem that needs to be solved in order to allow for region-based image indexing. Although the general approach settled upon is to use color, there are many ways to use this information to achieve segmentation.

In this chapter we explore methods of extracting information which have a biological basis. Because of this, concepts from cognitive science will be discussed. To facilitate this, a basic understanding of the workings of the retina and of the human visual system in general is assumed

throughout this chapter. Readers who desire background information can look up any introductory neuroscience textbook, for example [35].

The approach to filtering presented here is based on the definition of a general filter object which can be applied to an image at every image position. This idea is similar to the use of wavelets, except that the filters defined and used are much simpler in form. The concept, however, is derived from work in cognitive science. The human brain is known to continuously compute a variety of local comparisons in parallel across the entire visual input. The most commonly known comparison are the red-green and the blue-yellow color-opponent filterings, which are known to take place as early as at the retina. Luminance-based filtering is also known to take place, but it is not clear whether this is performed at the retinal or cortical levels.

## 5.1 Opponency of Values as a System of Measurement

As used here, color opponency is defined as the weighting of a value positively for the presence of one measure and negatively for the presence of another. When computing a given filter over a particular area, the result is the sum of the filter values times the subtraction of the negatively weighted measure from the positively weighted measure. Thus, the two measures being considered can be thought of as opposites.

Opponency is used in filtering for several reasons. The main reason is that this is a known feature of the visual system. It is also a more robust measure, since information is used for each local computation. If a region of color is encountered which has the excitatory color strongly represented but the inhibitory color not represented, then the filters will measure high values throughout, for any kind of feature shape. This is not the desired performance. Another reason is that this method is better computationally, as one computational pass can be used to measure two variables. The additional information present also decreases the effect of noise.

Very early on in the visual pathway, color opponency, of red-green and blue-yellow, is computed in a roughly circular center-surround manner. This is depicted in figure 5-1. The central region is preferentially excited by one input type, and inhibited by another input type, while the surround reacts in the opposite manner. Thus, flat regions of either input type have no effect. Instead, the locations of change in either type are selected. By having color opponency, changes between the two colors are made more significant than either color alone.

## 5.2 Types of Measurement

Several types of opponency are explored here. The simplest methods are the red-green and the blue-yellow color opponencies. A second method uses only hue value, in the HSV space, or the sum of all of the color channels, in the RGB space, as the measure being weighted. A third method considers the distance of the color from a given color, either by using the RGB or the HSV color comparisons.

For red-green color opponency, the input types are the red and green color values in the RGB color space. For blue-yellow color opponency, the input types are blue and the average of the red and green color values in the RGB color space. This is because when additively combining colors, which is done for example by television sets and computer monitors, red and green combined result in yellow.

## 5.3 Types of Filters

Several filter types can be deduced from the current knowledge of the response patterns of neurons throughout the visual processing stream. At the lowest level, response patterns which are maximal for spot stimuli are found. As early as the visual cortex elongated stimuli of a particular orientation begin to be the inputs resulting in a maximal neural response. Farther on, more complex features such as corners, stars, hand outlines, or particular faces are the maximal stimuli.

The current understanding of these response patterns is that at the lowest level neurons with small circular receptive fields compute the center-surround response at each location on the visual input. At the next level, these responses are then combined into elongated receptive fields, yielding neurons which respond maximally to lines which have a given orientation and are located at a given location in the visual field.

These edge detectors can then be combined into more complex stimuli. How a corner detector could be built seems fairly straightforward. How more complex detectors are set up in the brain is still unknown. A feature of the visual processing stream which may aid in creating more complex detectors is that every filtering is performed at several levels of resolution. Thin edges are detected at one level, wide edges are detected at another, and so on. By having multiple parallel levels of filtering, approximators which are more noise-free can be combined with approximators which have more noise but provide greater detail.

The computation required to perform this filtering can become large. Therefore, only the simplest filters were considered in this exploration. These filters are of two kinds: filters to detect spots, and filters to detect edges.

### 5.3.1 Spot Filters

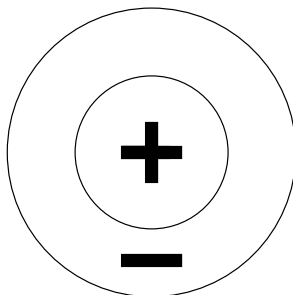


Figure 5-1: The simplest center-surround filter, a spot filter, has an excitatory center and an inhibitory surround.

Spot filters are fairly straightforward. The central region is excitatory and the surround is inhibitory. Spot filters emphasize local changes without providing information as to the directionality of the gradient.

In their implementation, spot filters were approximated by square arrays whose center contained positive values and whose surrounding area contained negative values. As the central region becomes larger, the measurement made becomes more robust.

### 5.3.2 Edge Filters

In the visual system, edges are described in terms of the combination of the outputs of the primitive circular center-surround sensors. This can be achieved by combining several center-surround sensors in a manner similar to that shown in Figure 5-3, by, for example, summing the outputs of these sensors. In the implementation used, edges were computed using a separate filter which has an elongated positive field buttressed by two negative fields.

Edge filtering can be performed in two different ways. Either localized or step-wise changes of a given directionality can be searched for. Localized changes can be searched for by using edge filters. Step filters, a variation on this theme, allow the emphasis of stepwise changes without focusing on a given line width. A step filter is equivalent to an edge filter which is cut lengthwise and then



Figure 5-2: The result of the application of a spot filter over two images. For some images the results are very good, while for others the results are very noisy.



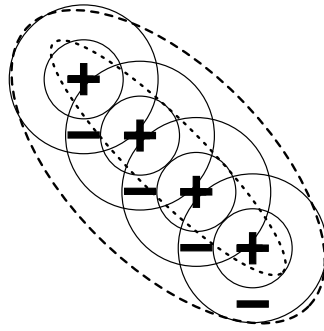


Figure 5-3: A center-surround edge filter can be created from the combination of a set of center-surround spot filters.

stretched to the original width.

### 5.3.3 Excitatory and Inhibitory Filtering Field Shapes

A modification to the simple filtering paradigm is the enhancement of the sought-for change through the emphasis of this change. This can be done by having the outer, inhibitory field have a very negative value next to the border with the excitatory field, and then increasing in value, thus becoming smaller in magnitude, as the distance increases. This results in an emphasis on abrupt local change as opposed to the smoother change which is emphasized by flat value fields.

The excitatory region, however, is always flat, as it is contained within the filter. If a similar shape were used, then a circular shape filter would result, where the true center of the filter would matter little in the shape being detected.

## 5.4 Output Modifications

The output of the application of any given filter at any given position in an input image is a single value. This value does not have a direct correlate to a visual representation. It is only a measure of the belief which can be placed in the existence of a given feature at this location in the image.

To make this output usable, several things can be done. First of all, the location itself can be tagged. Secondly, the entire filter image can be painted onto the locations where the filter is applicable. When using non-binary outputs, the filter, weighted by its local output, can be painted at each image location. Third, instead of painting the entire filter, only the feature which is being detected can be painted onto the image.

Tagging each location individually this is useful in determining where each filter is applicable in a given image. However, it does not aid in any image processing step, because at best this results in a series of pixels which are tagged with different values. Alternately, the filter itself can be painted onto that location. This results in the emphasis of continuous features as well as the inhibition of neighboring filters. Lastly, the entire area which the given filter overlaps when it computes a positive value can be tagged. This would result in the given area being labelled as being similar to the pattern being searched for. For more complex filters, this method could be used to locate areas of texture in the image. These areas can then be segmented away, either disjointly or conjointly with color information.

## 5.5 Difficulty of Filter Application

Although the displays of filter outputs provides information which is useful for human viewers, the extraction of information from them is nevertheless difficult. Due to constraints on time this topic was not studied further. However, the approach presented here, used in combination with the methods presented in Chapter 7, which describes the extraction of edge descriptions, may prove successful in providing a robust approach to shape extraction.

The other extension to the topic of filters is the computation of texture. For example, information could be gathered by edge filtering the different levels of the color mapping representation presented in section 4.7. This would provide an estimate of the Fourier transform of the image. This, in turn, is equivalent of the approaches most commonly used for textural descriptions.

## Chapter 6

# Image Segmentation

*Time extracts various values from a painter's work. When these values are exhausted the pictures are forgotten, and the more a picture has to give, the greater it is.*

Henri Matisse

The complexity of generalized image understanding lies in that several objects have to be recognized and segmented simultaneously during most interactions with the visual world. The difficulty lies in selecting which stored image to match with which part of the visual input, and to what extent the different views have to be combined in order to yield a match for an input. The issue of image segmentation has baffled scientists for decades, and no clear-cut solution has been proposed. Such a solution may never be found, since in humans there is an apparent overwhelming devotion of cortex in the brain to image processing, denoting a concentration of resources upon this task.

When dealing with static images, the lack of motion information exacerbates this problem. How to do this well is still an area of active research. Many solutions have been proposed, some of which make assumptions about the image, and others which feed parameters, such as the number of regions expected, to the segmentation algorithm being used. One commonly used approach in image retrieval systems is the use of Expectation-Maximization to determine the segmentation of the image based on color.

The approach described in this chapter develops a different methodology to achieve the goal of image segmentation. A novel approach was developed in order to explore the possibility for a parallelizable approach to image segmentation. More traditional methods are usually serial algorithms which perform well on a serial architecture but are not biologically feasible.

Toward this goal, first, the growing of a coherent image region from a single point in an image is developed. Then, a related algorithm is presented which provides a full image segmentation. Both of these algorithms rely solely upon color information for segmentation, because textural information is not implemented in Imagina. The base assumption made in all of these algorithms is that an image can be segmented into coherent regions based on the similarity of the color which makes up those regions.

## 6.1 The *Grow* Algorithm

The first algorithm developed for region segmentation is the growing of regions from a selected point outwards. From the starting point, each neighbor is considered in turn, and all neighbors which are sufficiently close to the value of the selected point are added to the region being grown. The region growth is performed in an outwards-moving fashion, such that all current neighbors are added to the region before any of their neighbors are considered. This is shown in figure 6-1.

The region being segmented is expected to result in contiguous points on an image plane. Although there are point-by-point variations due to noise in any image, this is not a problem. Once a region has grown beyond a few pixels, multiple paths will exist from the current region's boundary to a similarly colored point, unless another colored region is in its path. When starting, however, the local values affect performance significantly.

All of the points which are labelled as part of the region at the end of this algorithm are similar to the starting point by some measure. Whether to include a given point is based upon its similarity to the point where the *Grow* algorithm is initiated at or to a pre-computed value which is fed to the algorithm.

The similarity measure between points can be performed in two ways, based on the two color spaces which are used in Imagina, the RGB and the HSV color spaces, which are described in chapter 4. The similarity of two colors is computed by subtracting the distance between the two points from one. The RGB description provides for faster computation, while the HSV description gives a more acceptable color-based distance.

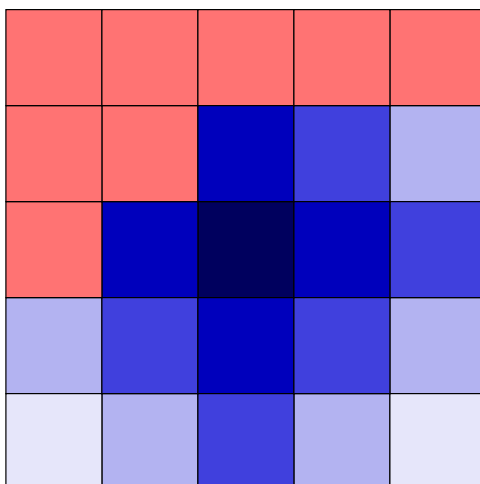


Figure 6-1: The *Grow* Algorithm. The algorithm starts from a single point, shown in black. The points which are similar to the first point are shown in blue, and the points which are dissimilar are shown in red. The first group of points added are in dark blue, the next set of points added are lighter, and so on. Only the points in blue are added to the region.

The algorithm is called *Grow* because point inclusion into a region starts at a given position and is expanded outwards. If a given point is included, then all of its unmarked four neighbors are considered for inclusion. If a given point is not included, its neighbors are not considered for inclusion into the region, either, unless they are neighbored by an included point on a different side.

In order to determine which points to include, their similarity to the comparison value has to be measured and compared against a cutoff value. This is because regions are defined by an all-or-none labelling. A point is either inside of the region or outside of the region. The best cutoff value depends on the color space as well as on the region being grown. Although a value which provides a perceptually coherent segmentation in most cases can be selected, there is no value which works well in all cases.

The main weakness of the *Grow* algorithm is that a particular starting point, a particular color cutoff, as well as a color space, must all be selected. If the starting point is not close to the average of a given region, the *Grow* algorithm will fail to find that region. Two extensions, the *Adaptive Grow* and the *Progressive Grow* algorithms, attempt to address some of these issues. The performance of all three algorithms is shown in Figures 6-3, 6-4 and 6-5 starting on page 64.

## 6.2 The *Adaptive Grow* Algorithm

To lessen the bias imposed by the value of the starting pixel, an average of all of the points which have already been included into the region can be kept. As more of the neighbors of the starting pixel are included, the value being compared against becomes a better approximation for the average value of the local region. Even if the starting pixel is not very representative of the region, in most cases this algorithm will succeed in segmenting a cohesive region.

Again, a cutoff value has to be selected. However, this value usually can be greater than the value needed for *Grow*, because in most cases the average of a region is a better comparison than a single starting point value. This algorithm is identical to the previously discussed *Grow* algorithm. Because of its adaptive nature, however, it performs better at segmenting image wholes.

## 6.3 The *Progressive Grow* Algorithm

Instead of having a single global region average, a local average can instead be kept. This average is progressively modified based upon the most recently included points on every outward path from the starting point for region growth initiation. Thus, *Progressive Grow* can adapt to the local area around every point of the region instead of depending on a single global average. This can be useful when a region contains a low gradient from one side to the other. Human vision handles such situations without the imputation of region boundaries at several locations. This provides support for an algorithm with a progressive average.

The algorithm, shown in Figure 6-2, has a local average kept for every point on the outer hull of the region as it is being grown. The neighbors of each point are considered for addition based on this local average instead of a global average. A point which has two neighbors which are currently a part of the region is compared to both of them in turn. Because progressive averages for nearby pixels are usually very similar, computing an average among neighboring values before considering a point for inclusion is unnecessary.

The updating of the local progressive average is fairly straightforward:

$$\text{new average} = (1 - \text{weight}) \cdot (\text{old average}) + \text{weight} \cdot (\text{point value}) \quad (6.1)$$

where the starting **old average** is set to be equal to the value of the point where the *Grow* *Progressive* algorithm is initiated at.



center of the skirt, and the cutoff and algorithm applied are varied.



Figure 6-3: The sample image being segmented and the basic *Grow* Algorithm's performance on it. The cutoff values are .6 and .7, top and bottom, for the segmentations performed on the right.

## 6.4 Methods for General Segmentation

The problem of performing general segmentation is more difficult than the simple segmentation of each individual image region. Even if a set of parameters and a version of the *Grow* algorithm is settled upon, selecting the starting points for region growth and sorting out conflicting region assignments can be difficult. There are two approaches to the issue of general segmentation: working on finding good local starting points, or attempting to perform the segmentation at a global instead of a local level.

### 6.4.1 Using Saliency for Segmentation

When segmenting an image, it is hard to tell where the best place to segment an image is. None of the usual approaches to this problem are very reliable. Using a color histogram range based on the color content of the image can fail by splitting a region into two bins, or by combining several disparate regions into one. Segmenting areas based on local averaging computations can also run





Figure 6-4: The performance of the *Adaptive Grow* Algorithm with cutoff values of 0.84, 0.90, and 0.94. The HSV color space comparison is being used.



Figure 6-5: The performance of the *Progressive Grow* Algorithm using HSV color similarity on the left, and RGB color similarity on the right. The new pixel weighting for averaging is one fourth in both cases. The cutoff values from top to bottom for the HSV similarity are 0.84, 0.90 and 0.94, and for the RGB similarity are 0.76, 0.84 and 0.90.

into difficulties as the average being used to add new pixels changes with the starting location of the segmentation process. This latter process also is somewhat questionable as it relies on an iterative process which cannot be parallelized.

One approach to using saliency is discussed by Itti, Koch and Niebur in [15]. Although their model of visual saliency is designed for visual attention, we will argue for its applicability towards image segmentation. One can argue that the sections of the image which appear most salient to the viewer will be most easily remembered. Therefore, if a measure of saliency similar to humans' can be computed, the sections of the image which are most salient by this measure should be segmented preferentially.

Itti et al. use “center-surround” operations to compute for the same features on the same image when viewed at different sizes. Red-Green and Blue-Yellow center-surround values are computed, as well as directional filters for four orientations. These computations mimic the computations found in the retina, laterate geniculate nucleus, and visual cortex.[35][7] This approach is also explored in Chapter 5 of this thesis.

The difficulty with these computations is that combining the different value maps is not straightforward. To avoid the overcrowding of data with noise, the maps which have very few peaks in them for a given input are weighted more heavily in determining saliency of locations. This is measured by comparing the highest peak with the average of each map, and is based on neural lateral inhibition mechanisms in the visual cortex. Different maps in the same modality compete, while different modalities contribute, to the overall saliency map.

The computed saliency can be used to determine the locations to begin the image segmentation. Thus, the parts of an image which are most likely to have been foveated by human viewers, and therefore the parts which are most likely to be remembered, are most likely to be segmented away. Thus a higher rate of image matching based on user queries is expected than if the starting segmentation points were to be selected randomly.

#### 6.4.2 Using Hierarchy for Segmentation

A hierarchical approach to segmentation bypasses this issue. Instead of selecting points, the *Grow* can be performed at different levels of image resolution. The region labellings can then be propagated from the lower resolution image versions to the higher resolution image versions.

Starting locations never have to be selected because the growth is performed at all points

simultaneously. A comparison cutoff, as well as a color space, must still be selected. The advantage over a hierarchical approach, however, is that the serial nature of the *Grow* algorithms which have been presented is overcome.

This approach is not perpendicular to the method of using saliency. For example, the concept of saliency could be applied to constrain the growth of regions. Regions attempting to cross saliency boundaries could be stopped or redirected. Although such a combined approach may be fruitful, it was not pursued here.

## 6.5 A Hierarchical Recursive Adaptive Region Segmentation Algorithm

The algorithm settled upon for image segmentation in Imagina performs segmentation hierarchically. It relies on the Color Mapping representation to provide a description of an input image at different levels of resolution. The Color Mapping is segmented recursively, with the segmentation determined for a given level propagating to the level below. The final segmentation is then applied backwards to the original image layout. In this discussion, the ‘highest level’ refers to the level of the Color Mapping which is the smallest, containing only 4 points, and the ‘lowest level’ refers to the level of highest resolution of the Color Mapping. The point at a higher level whose value is the average of four points at a lower level is called the ‘parent’, and the four points the ‘children’.

The algorithm starts at the highest level of the Color Mapping representation, by assigning all points which are similar by a given metric to the same region. Up to four regions can be initiated at this step, which can happen if all of the starting points are dissimilar from each other.

The iteration is performed in two parts, along with several performance optimizations which will go unmentioned. The labelling at a given level propagates downwards to the next lower level by having the four pixels which are represented by a single pixel at a higher level be considered for inclusion into the same region that the higher level pixel is a part of. This is shown in Figure 6-6.

Because the average against which points are compared against is the global average of the region to which the parent point has been assigned, the algorithm is an adaptive algorithm. Although the mean of the regions can change between levels due to the variation of the points included in the region, the variation is not performed locally at each point inclusion, as would be the case in a progressive averaging algorithm.

After the labellings are propagated, some points at the lower level will be left unassigned. These

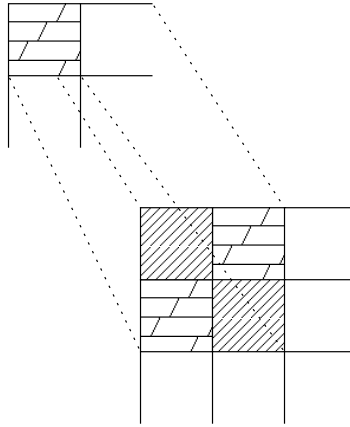


Figure 6-6: The propagation of labellings from the top level downwards. The four points which are represented by one point at a higher level are considered for inclusion into the same region as their parent. The similarity is measured against the average of the region to which their parent belongs, not to a local average, such as the parent's value.

points are either joined with other unassigned neighbors to which they are similar into a new region, or are assigned their own individual region label.

Lastly, the entire image array at the given level is passed through, and every neighboring pair of regions is compared for similarity. If they are sufficiently similar, the two regions are joined into one. This step guarantees that regions that are elongated and therefore not visible at higher levels are nevertheless found and joined into one at some level.

This algorithm also needs a cutoff value for determining point similarity. It also requires a cutoff value for determining region similarity. Although intuitively it would seem that the region similarity should have a higher cutoff, performance improves for cutoff values which are approximately equal for both region and point similarity determinations. This algorithm, however, provides usable segmentations over a large range of cutoff values.

### 6.5.1 The Algorithm as a Feedback Approximation

This algorithm is an approximation of a multi-level algorithm with feedback, which would assign labels at each level individually, and attempt to 'convince' adjacent levels of the reliability of the labelling. Initially, the labels could be arrived at randomly as much as from local information. If the labellings are random, then the four points at a lower level are unlikely to be coordinated enough to influence the labelling at a higher level. This can happen reliably only if the information

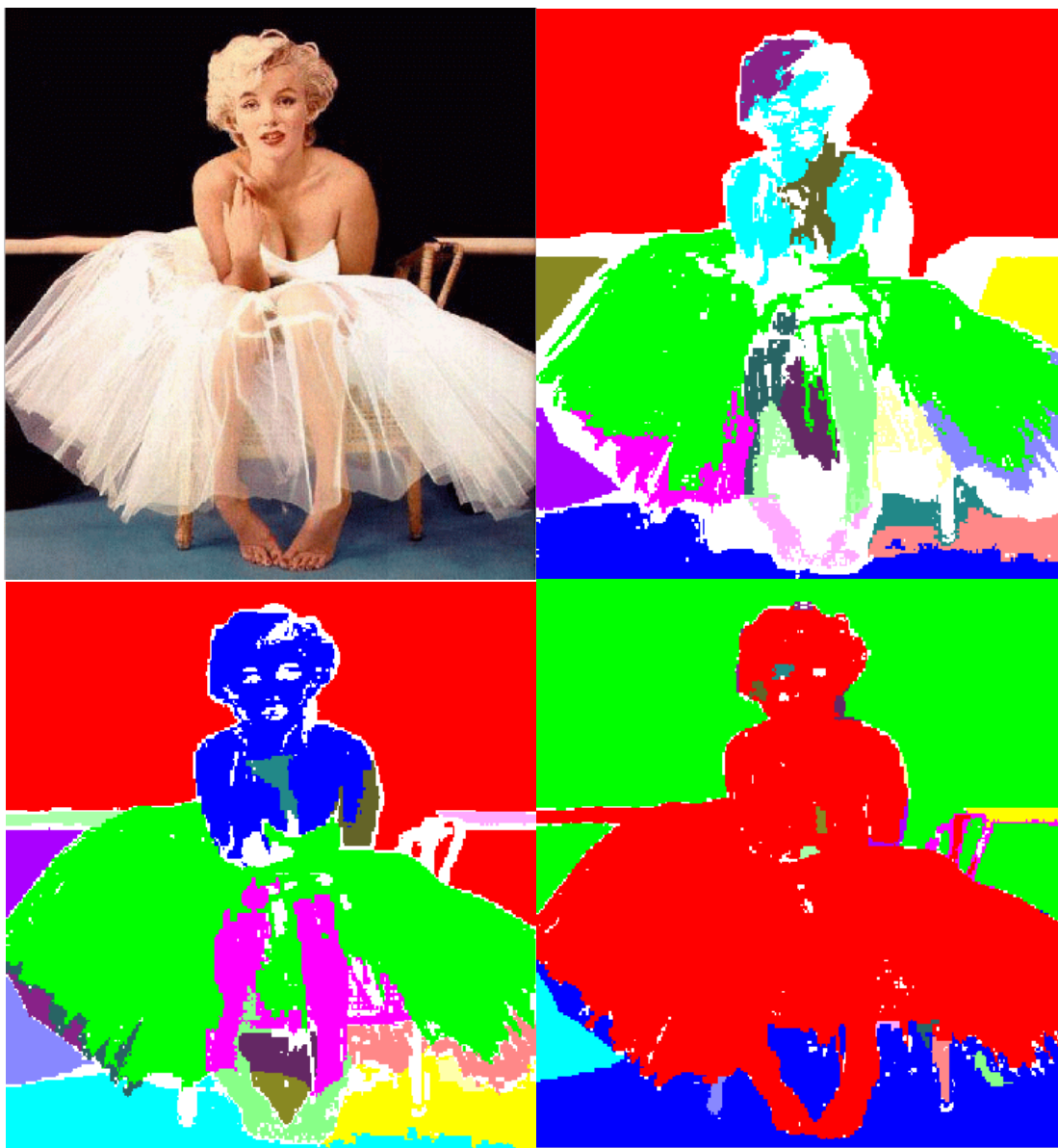


Figure 6-7: The algorithm applied to the same image previously segmented with the *Grow* algorithms. The segmentations shown use cutoffs of 0.94, for the one next to the original image, 0.90, for the one below, and 0.80 for the third segmentation. The segmentation using a cutoff of 0.90 is the segmentation which the Imagina system defaults to upon loading this image. The color labels displayed are simply color markers used to visually denote the twenty most populous regions which are segmented by the algorithm, and have no intrinsic meanings.

at both levels support the labelling. Thus, the information which is acquired locally is combined with information which propagates from other levels.

### **6.5.2 Extensions**

The algorithm could be improved by applying statistical information to the combination of means. This can be done in two ways, either by considering the statistical reliability of the point values at high levels of the Color Mapping, and which therefore are averages over large regions of the image being segmented, or by considering the statistical reliability of the mean of a given region which has been created and whose mean may represent only a few or many points.

In the former case, the pixel similarity required for joining points into a region would be higher at a higher level of the Color Mapping than at a lower level. In the latter case, the pixel similarity required for joining points into a region would be very low for regions containing only a few points. However, as the regions would grow beyond a handful of points, the cutoff similarity would increase. The difficulty with this approach is that some points may be included into a region without belonging into the region after the range of inclusion becomes limited.

## Chapter 7

# Boundary Edge Extraction

*There is a boundary to men's passions when they act from feelings; but none when they are under the influence of imagination.*

Edmund Burke

After an image has been segmented, it consists of a set of one or more regions. Each region is fully connected and contains pixels which are similar by some metric. The goal thus becomes to come up with a description for the shape of that region. Different shape descriptors are possible, all of which operate on a set of edges describing the boundary of a region. This chapter describes how these boundary edges can be extracted. The geometric descriptors that build upon these edges will be the topic of the next chapter.

The edge extraction methods developed here assume an input region which is fully connected. Moreover, its robustness relies on the input region being full. If holes are present, they can be filled as a preprocessing step to edge extraction. The goal of the edge extraction step is to construct a connected set of edges that describes the boundary. This edge description can then be used to compare region boundaries by different methods, as described in chapter 8.

The purpose of edge extraction is to take the set of unordered points that form a region, and construct a more meaningful representation that describes the region. From a pixel-based representation, we would like to go to an edge-based representation. It is easier to compute with



edges, since they compress the information contained within the set of input pixels to a more manageable form.

Edges are certainly only an approximation to the set of raw pixels. However, such a loss of detail is inevitable when a processing step abstracts away from the raw image data. When constructing the edge representation we introduce a bias to the information contained in the raw pixels due to our choice of a particular algorithm to find the edges. However, finding the edges distances us from the raw pixel information and brings us closer to an abstract representation of an object.

## 7.1 Prior work

The edge extraction is a geometric processing step. Its purpose is to take a set of pixels, and turn them into a more meaningful edge representation. Previous work comes from the computational geometry community.

### 7.1.1 The *Crust* Algorithm

In computational geometry, a solution to a harder and more general problem has recently been proposed. Amenta, Bern and Kamvysselis [1] published an algorithm that finds the shape of an object in three dimensions from sample points in its boundary. The algorithm is based on the Voronoi diagram of a shape to find an approximation of the medial axis of the shape (see figure 7-1). Based on that medial axis, the algorithm connects neighboring points that do not cross the medial axis, the exterior of the object, its “crust”. This algorithm is simple, fast and robust, and it reconstructs a provably correct linear approximation of a shape that meets the sampling criterion. Moreover, it only needs a set of unordered points to work upon.

To use the Crust algorithm, however, a preprocessing step is required on the points of the image region in question, that finds all the points on the boundary of the shape. The next section presents an algorithm for finding those points on the boundary. Section 7.3 goes a step further by using the extra knowledge that we have of where the inside of the region is, and reconstructs the boundary edges directly, without use of the Crust algorithm.

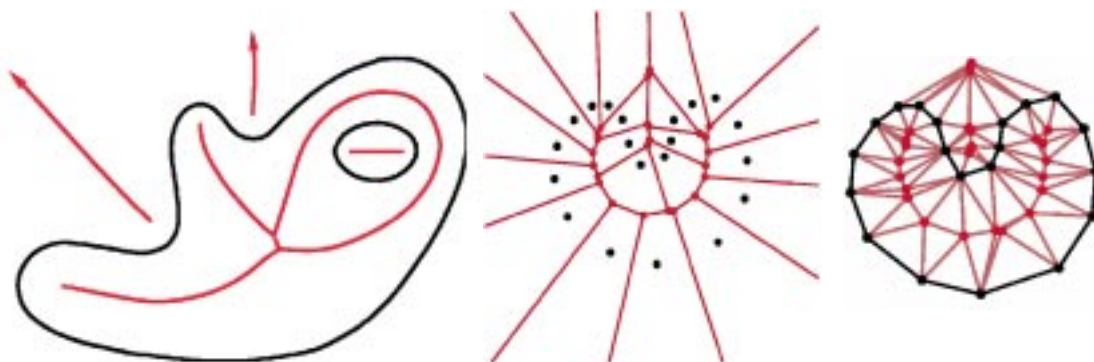


Figure 7-1: Two-dimensional *Crust* Algorithm. Left: The medial axis (in red) of a figure is the set of all centers of circles inscribed in the figure and touching its boundary in at least two points. Middle: The Voronoi centers (red points) approximate the medial axis. Right: Joining the nearest neighbors without crossing the medial axis gives an approximation to the shape.

## 7.2 Pixel-Based Boundary Extraction

Several straightforward approaches exist which allow the determination of the boundary of a region after it has been segmented from the image. These are all based on processing the region on a pixel-by-pixel level. Although these methods are easy to implement, they run into difficulties with noise and special cases.

The simplest way to find the boundary of a region is to pass a window of 3 by 3 pixels over the image, and label any center of the window as part of the boundary if the center is part of the region and any other pixel in the current window is not part of the region. The resulting set of pixels can then be sorted and connected, or walked over in a method which can detect loops.

Another approach to this problem is to circle the boundary of the region starting from a point which is known to be on the boundary. Such a point can be found based on the center of mass of the region. Given that the region is fully connected, that is, every pixel is reachable from every other pixel within the region by passing through only region pixels, some pixel must have the same x or y coordinate as the center of mass. Checking all y or x values along either of these coordinate values, respectively, will yield a pixel location.

This starting position can be picked to be either on a left or a right edge of the region, by either starting from the left or the right of the center of mass location. The direction of movement is always forward with the region on the right; thus, starting on a left edge the direction of movement

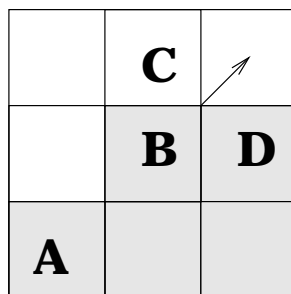


Figure 7-2: The local pixel-based boundary growing algorithm. Pixels within the region are shaded, while pixels outside of the region are not. The current position is at **B**, and the current direction is shown by the arrow. **C** is the first position considered for the next step, and **D** is the next position to which the algorithm will step to.

is upward, and on the right edge it is downwards. Based on the starting position and direction, the current position is iteratively updated by stepping one pixel into the current direction.

Before stepping forward, however, the current direction has to be updated. It is tentatively turned leftward by 45 degrees. The positions around the current pixels are then iteratively attempted, going in order in a clockwise fashion around the pixel. This step is shown in Figure 7-2. The position is then updated again, with the direction of this pixel from the current location. The previous position is always the position in the opposite direction of the current direction. The next position is always a pixel within the region whose boundary is being determined. The direction at the end of a step is always in the direction of the next pixel to be considered.

In most cases, this method will succeed in describing the region boundary. This method, however, has several downfalls. Noise affects the boundary which is returned greatly. Single-pixel connections between parts of a region also cause problems, requiring constant back-stepping and a renewed attempt at edgewalking from a previously saved position. Although the base implementation is trivial, a fully correct implementation can become very complex.

The single advantage of this approach is that it provides a complete description of a region boundary. From the list of pixels, any desired boundary description can be defined, because none of the information, or noise, has been removed.

## 7.3 Window-Based Boundary Extraction

The fact that the shape is filled to go a step further than the previous boundary walker, and construct a new edge extraction algorithm. This algorithm considers a larger neighborhood of points at each iteration, and guesses a boundary edge for each such window of computation.

Increasing the window size improves the reliability of the algorithm, as well as its robustness against noise. Moreover, instead of points on the boundary, this algorithm constructs directly boundary edges, so no postprocessing step is necessary.

At each step, the center of mass is computed for the ‘inside’ and ‘outside’ points within a window of pixels. This information is then used to deduce the outer edge of the region being segmented, as well as the direction of this edge. Edges are then combined into a perimeter, which is then smoothed to remove multiple edges which are neighbors and nearly parallel.

Although the construction of the algorithm is serial, it can be easily parallelized by placing the processing window as a filter over every location of the image, and getting boundary edge approximations for each of window. Multiple scales can be run in parallel, and the different layers can then converge into a single multiscale boundary determination. The edge smoothing step, being an approximation of the weighted interaction of multiple levels of detail, would not be required in this case.

### 7.3.1 Algorithm Overview

The window-based boundary extraction algorithm finds tangents to the extracted region at different points around it, and then constructs a piece-wise linear description of it. The tangents are calculated on a window which contains at least one point of the region. Making the computation window size larger results in a less precise but more robust edge description. A smaller window size will pick out more detail, but will sometimes overfit a shape, by describing the noise which results from the imperfections of the color segmentation phase.

The more tangents that are taken on the boundary, the more correct the description can be. However, computation will be wasted for possibly similar nearby tangents and redundant information. Similarly, a description can include hundreds of edges, or simply a few dozens. Nearby small edges can be combined into larger ones if their slopes are similar. Again, the tradeoff exists between the precision of many but almost identical edges, and fewer larger but less precise edges in terms of the noise robustness and processing time cost.

### 7.3.2 Serialized vs. Parallel Algorithm

The algorithm described is implemented as a serial algorithm, for simplicity and efficiency on a single-processor machine. The parallel algorithm requires much more computation overall, but drastically reduces the computation required of any single processor. In the parallel version, every processor is responsible for finding a single tangent at a single location for a single window size. In the serial algorithm, one tangent is found at a time, for a particular location and a particular window size. To optimize the algorithm the direction and position computed for one tangent are used to determine a good placement for the next window of computation. In a parallel version of the algorithm, several window sizes would be applied independently at each image location. We will not describe further the parallel algorithm. Instead, we will concentrate instead on the actual implementation of its serialized counterpart.

### 7.3.3 Computation within a Window

Within each window we calculate the center of mass of the points belonging to the region, and the center of mass of the points outside of the region. The perpendicular bisector of the segment joining the two centers gives the direction of the edge. The relative sizes of the 'inner' and 'outer' regions within the window give the position of the edge. Moreover since we know which center represents the inside and which the outside of the region, we can construct directed edges for the boundary.

#### Choosing the window size

The window size is an important parameter for the algorithm to work. If a shape is large and free of holes, then a large window size works most of the time. The few times a large window size is inadequate is when the region has sharp corners, where a smaller window size is needed. A smaller window size can be chosen for objects that contain narrow parts, such as the picture of a hand with fingers spread. When the region is thinner than the window size, the window size should be reduced.

#### Increasing the window size

When the center of processing is either too far in the region and there are no outside pixels within the window size, or when it is too far outside the picture and there are no region pixels within the window size, then there is no information on how to proceed. The tangent cannot be calculated

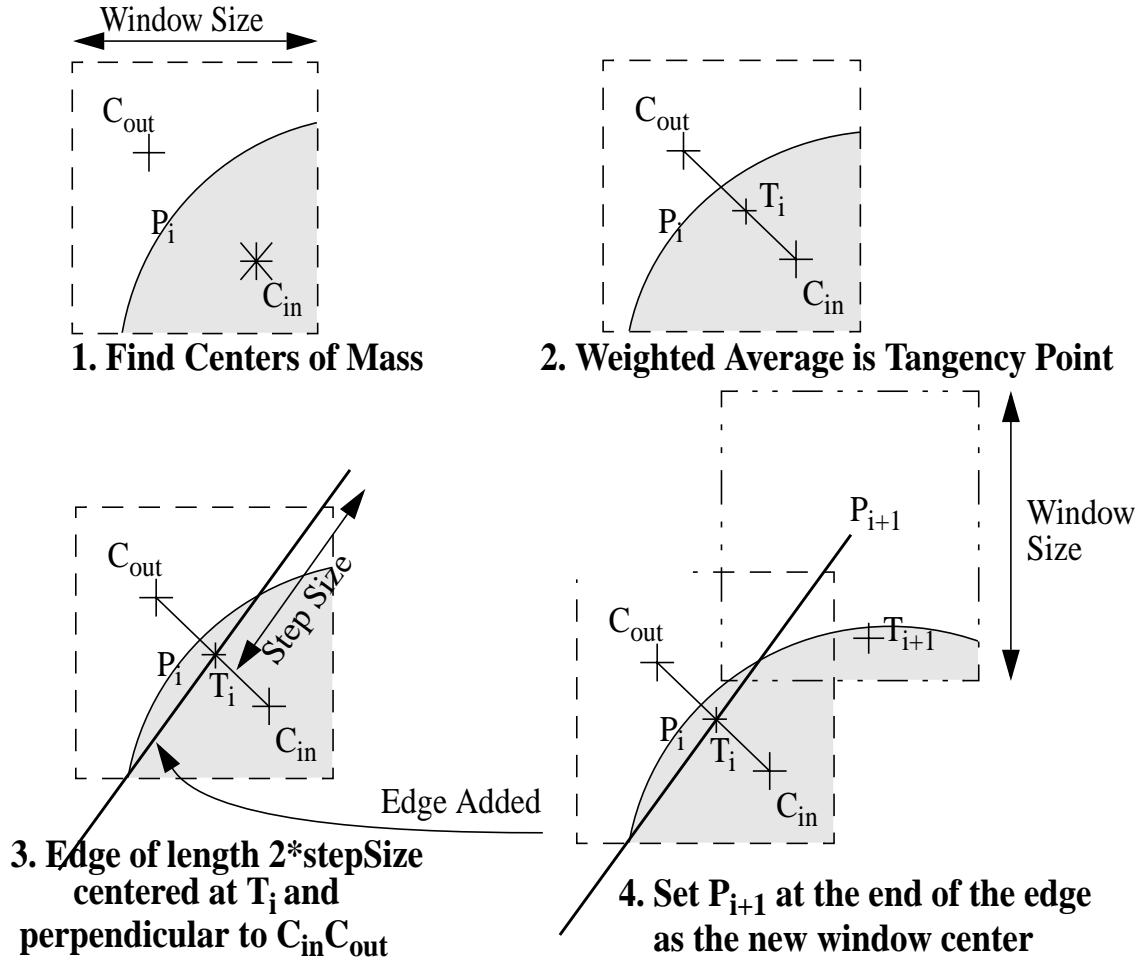


Figure 7-3: The Edges Boundary Extraction Algorithm finds an approximation to an edge within a window of computation. It calculates the center of mass  $C_{in}$  of the points belonging in the region and the center of mass  $C_{out}$  of the points belonging to the outside of the region within the window of computation. It uses the number of points belonging to the inside and the outside to calculate a weighted average of  $C_{in}$  and  $C_{out}$ , that will serve as an approximation to the tangent point to the region's boundary. A tangent direction is calculated to be parallel to the perpendicular bisector of  $C_{in}C_{out}$ . Finally, a new window center is calculated by moving along the tangent.

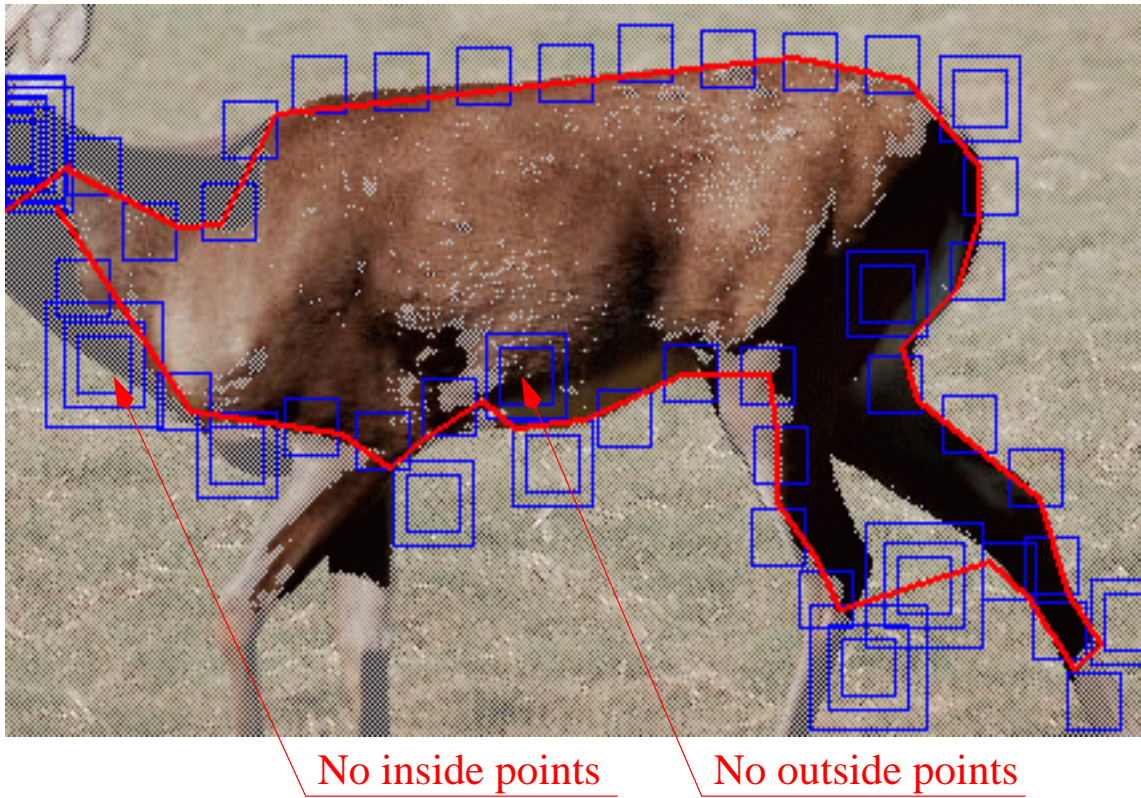


Figure 7-4: Increasing the window size: When there is not enough information within a window for calculating an edge, the window size is increased until enough information is present. This figure exhibits two cases where this situation arises. Where there are no inside points or there are no outside points, the window doesn't have enough information to place an edge, since only one center of mass can be calculated. A more subtle case arises when the two centers of mass coincide, not shown in the figure.

and the progression is stuck. In that case, a simple solution is to increase the window size until both inside and outside points are included with the window of computation. Once the boundary is found again, the window size can be restored at the next step. Another case when there is not enough information to determine an edge is when the inside and outside centers coincide. In that

### **Placing the window of processing**

As described above, in a parallel algorithm where large numbers of parallel resources are available, a window can be placed at every point in the image and obtain a lot of tangents that can be joined to construct an object.

On a single processor, we should limit the number of such computations. Therefore, a window is placed only at points on the contour of the region, where part of the window contains pixels inside the region, and part of the window contains outside points.

The first window is placed by simply following a line from the outmost point of the image towards the center of mass of the region. When a pixel belonging to the region is encountered, the center of the window is placed.

Once a tangent has been found within the current window, subsequent windows are centered by moving along the direction specified by the tangent. The process is then repeated until the edges have gone full circle returning to the initial starting point, or until a certain maximum number of iterations is reached.

### **Step Size vs. Window Size**

To determine how far to move on the direction of the tangent before placing the center of a new window of processing, we use one more parameter, the *step size*, which determines the absolute distance between two consecutive window centers.

The two notions of window size and step size are related but not the same. The window size determines how many pixels are going to be used for the computation of a particular tangent. The step size determines how many tangents are going to be computed. Together, they determine how much computation will be spent on extracting the edges of a particular region.

Several groupings of window and step sizes can be considered:

- With a large window size and a small step size, many certain steps are taken in going around the region. This method is going to ignore noise, as well as detail on the shape boundary.



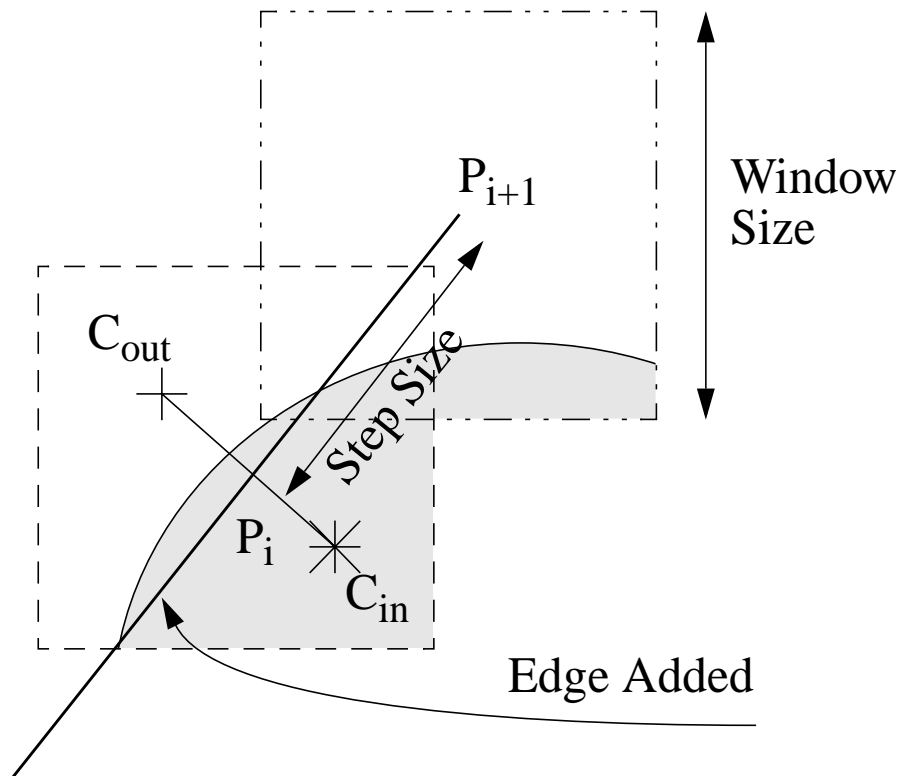


Figure 7-5: The difference between window size and step size. The window size determines how much computation is performed at every iteration. The step size determines the frequency of sampling around the boundary.

This is the most expensive computation.

- With a small window size and a small step size, the algorithm will pick out the most details, but also be prone to error.
- With a small window size and a large step size, the algorithm will be very likely to overshoot because of a small deviation in the boundary. It will be misled by little twitches in the region boundary which could be due to noise.
- With a large window size and a large step size, the algorithm will find few edges, but each of them will have enough information to make a noise-tolerant decision on what the boundary position and orientation are. By ignoring noise though, this method also ignores possibly useful detail.

There is no golden rule on how to make these value choices. Most certainly the same region will need different window sizes at different parts of its boundary. A simple way to achieve this effect is to have a default window size corresponding to the desired level of detail, and changing that window size as the conditions around the region change. When only inside or only outside points are found within the window, the window size should be increased. When either the inside points or the outside points fail to form a connected subregion within the window, the window size should be reduced after moving the center closer to the boundary.

### **7.3.4 Determining a tangent**

Once the window has been placed and the window size has been chosen, we can compute a tangent for the region boundary at that part of the image. The direction of the tangent, along with the position of the region boundary, together determine a straight line where an edge will lie. Using the step size we can set a length on that straight line, and thus determine an edge, whose midpoint is the point we found on the boundary.

#### **Finding a direction for the tangent**

Within the given window of computation, we would like to determine a single edge. To do so, we need a direction and a midpoint. The optimal direction is tangential to the region. This tangent is perpendicular to a vector that is easy to approximate.



## Window Size and Edge Re- construction

Figure 7-6: Decreasing the level of detail. A higher window size causes a greater tolerance to noise, and provides a rough estimate of the shape. A smaller window size allows more attention to detail, but also generates lengthier representations that might be harder to work with and might be overfitting noise from the segmentation.

We can compute the center of mass  $C_{in}$  of the points inside the region, and the center of mass  $C_{out}$  of the points outside the region. The vector  $\overrightarrow{C_{in}C_{out}}$  is a good approximation for the normal to the boundary within the window. The direction of the tangent is simply perpendicular to that vector.

### Finding a point on the boundary

Now we have to find a point on the approximated boundary between the inside and outside points. One way of finding such a point would be to move on the line that connects  $C_{out}$  to  $C_{in}$  until a point in the region is found. This method is inadequate because it will trust a single pixel of the region which could be the result of noise. More importantly though, it is inadequate because it goes against the spirit of approximation that the window method was developed in. We are really not looking for a perfect tangent to the region. We are looking instead for an edge representative of the region boundary in the particular window of computation. Therefore, instead of relying on individual pixels, we rely again on the two representative points that we found.

Since  $C_{out}$  is a representative of the outside of the image, and  $C_{in}$  is a representative of the inside of the image, the boundary should lie somewhere between the two. We could choose the middle of the segment joining the two centers, but we can do a little better than that. We can weight this average of  $C_{in}$  and  $C_{out}$  by the number of pixels belonging to each region. Thus, we can a formula for  $T_i$ , an approximation for the boundary point.

$$T_i = (C_{in_i} * Numpoints_{in} + C_{out_i} * Numpoints_{out}) / (Numpoints_{in} + Numpoints_{out}) \quad (7.1)$$

### Adding an Edge to the set of edges

Once a direction and a boundary point is set, we simply add an edge to the set of edges, centered at the boundary point with a length of  $2 * stepsize$ . Since the next center will be  $stepsize$  away, such a length is a good choice for a representative edge. The length of the edge is not that important since an edge's endpoints will later be extended or brought closer until the intersections with the two neighboring edges.



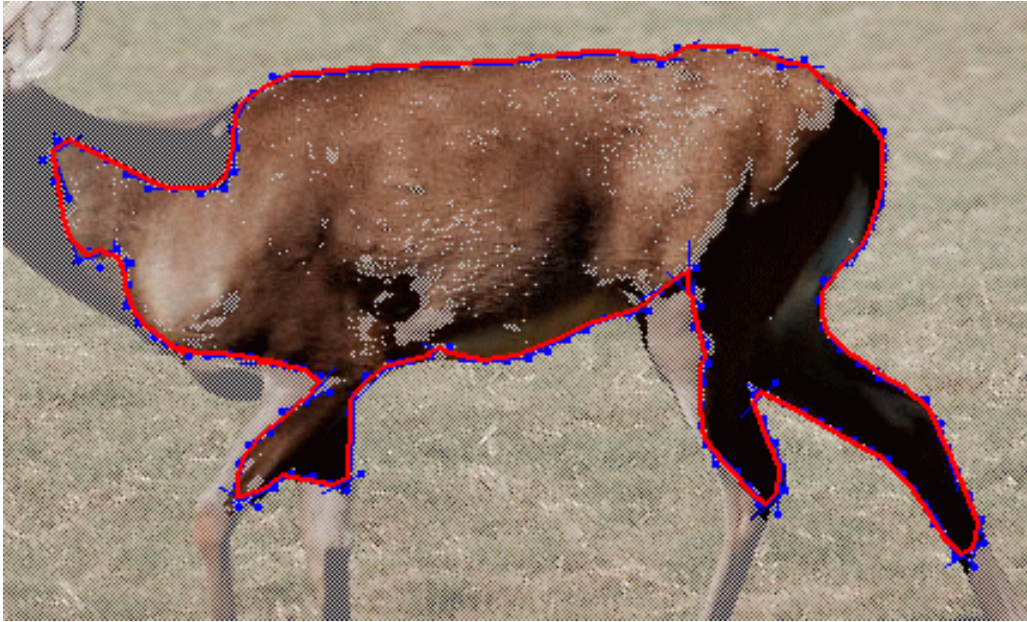


Figure 7-7: The complete boundary at the highest level of detail. A moderate window size and a small step size pick out the most detail from the shape

### 7.3.5 Post-processing on the Edges Found

Once those edges have been added, they undergo two stages of postprocessing before they can be passed on to the next stage that will use them to construct shape descriptors. This post-processing gets rid of the useless tips of intersecting edges, and compresses the representation by combining small consecutive edges that have similar slopes into larger edges.

#### Intersecting the Edges

The edge intersection is easy, since by applying a sequential algorithm the edges are already ordered by circling around the region. In the parallel form of the algorithm one would have to determine which edge to intersect with which other edge. In the serial case however, the choice is obvious. We simply set the endpoint of edge  $i$  to the intersection of edges  $i$  and  $i + 1$ , and the starting point of edge  $i + 1$  to that same intersection.

A complication arises when the two edges are nearly parallel. In that case the intersection falls too far out, and setting the endpoints to meet at infinity might not be the right decision to make. Therefore, we constrain the intersection of two edges to fall within the bounding box surrounding

the two edges. If the intersection falls outside that bounding box, then the midpoints of the two endpoints to be modified is chosen as the new value for both endpoints. After this step, the region is surrounded by a piecewise linear continuous set of edges.

### Joining similar edges

Once the edge endpoints are made continuous, that is once the end of one is the beginning of the next, a compression step can be carried out. One can make the edge representation of a region more compact by replacing small consecutive edges by larger ones if the small edges are nearly parallel.

This step is not a fundamental part of the algorithm, but can reduce the number of edges in the representation by a large factor. In our tests, tenfold reductions of the number of edges is not uncommon. The resulting loss in detail is negligible when compared to the much smaller number of edges.

One more parameter, *cosineCutoff*, specifies when two consecutive edges should be joined. The joining is simply based on the angle between the two edges, and never on the length of either edge. If the cosine of the angle between the two consecutive edges is larger than *cosineCutoff*, the two edges are joined.

When two edges are joined, the begin point of the first and the end point of the second are kept unmodified. A larger edge with those endpoints is created and replaces the previous two. Begin and end point can be determined simply since edges are oriented, circling the region counter-clockwise.

A simple version of the algorithm can be made serial, going around the region, and combining every similar edge with a progressively growing first edge. The disadvantages of this method is that first, it can't be implemented on a multiprocessors, and second, its result will depend on the order in which we circle around the region. Moreover, this version will have a tendency to growing some edges more than others, resulting in an unbalanced description.

Instead, we implemented an algorithm that is more easily ported on a multiprocessor. It compares neighboring edges two by two and combines similar ones into larger edges. It then compares the combinations two by two, and again combines similar ones. This hierarchical method has the advantage of being independent of starting point or direction of rotation. It has the advantage of being easily implementable on a multiprocessor. Finally, it has the advantage of generating progressively less precise descriptions of the object, that can be used for matching at

different levels of detail.

Independent of the algorithm that implements it, the step of combining similar edges into larger ones has the advantage of making the final description less dependent on the step size. Therefore, a smaller step size will pick out more detail if it's available, but will represent large straight sides compactly, by replacing them with a single edge. The step size can therefore be made smaller without risking cumbersome descriptions and having to later match a very large number of edges.

### **7.3.6 Improvements and Extensions**

Several extensions are possible to this algorithm. The first two are elaborations on the current processing methodology. The last two provide a different way of thinking about the algorithm and change its function in general.

#### **Decreasing the window size**

The major flaw of the current algorithm is that it doesn't have a way of detecting when the window size is too big. A simple way to get around this problem is to always start with a small window size, and augment it as needed. However, for completeness we could extend the algorithm to detect when the window size is inappropriately big for the shape to be extracted. This situation arises when the window contains more than one surfaces of the region. This is the case in crevices in the figure, where the full detail of a crevice will rarely be explored. This situation arises also in thin objects, that cut diagonally through the window size. In both of the above cases, the averaging will not yield an appropriate measure of the boundary edge position. One way of dealing with this problem is to draw a line between the outside center and the inside center, and observe how many times the line crosses a boundary of the region. If this number is higher than one, then clearly the shape is a stripe through the window, and the window size must be reduced. The new smaller window must be centered closer to the surface otherwise it might not include any region pixels, in which case its size would have to be increased once more. To move the center closer to the region, one can simply follow the line from the outside center to the inside center and set the new center where the line intersects the region.

## **Changing the Shape of the Window**

Another simple fix is to change the window size to be more rounded than the square currently used. A square is certainly easy to deal with, but it doesn't give equal importance to all points at the same distance from the center of computation. It will consider points along the diagonal that are further away than points along the vertical and horizontal axis. Along with considering a circle as shape of the window of computation, we can furthermore change the absolute cutoff of the window size for considering pixels in the computation. Instead we could assign an importance metric to each pixel, depending on its distance from the center of computation, and calculate a weighted average instead of an unweighted one on the pixels of the fuzzy region.

## **A novel similarity metric for region membership**

Moreover, instead of having a discreet value that is either zero or one, one can assign a floating point number that can take all values in between. The labels would thus be replaced with a computation every time the pixel is accessed, that evaluates a similarity metric between the pixel being currently processed, and the average of the region it belongs to. This similarity metric can be based solely on color, texture, shadowing, or other criteria. It should return a value between 0 and 1 however, where 1 means the pixel belongs without doubt to the region, 0 means the pixels definitely doesn't belong to the region, with all possible values of certainty between the two.

When the centers of mass of the window are being computed, certain pixels will only contribute partly to the calculation, instead of either belonging or not belonging to the region, thus achieving a greater precision in the calculation. This extension in fact would make the region extraction step obsolete. The edge extraction algorithm could be directly applied to the raw image pixels. To determine the average color of a region however, a filter must be applied to the image to find patches of similar colors. This would be a much simpler form of the current segmentation mechanism described in chapter 6. However, since the segmentation functionality is already provided both globally and locally with the grow algorithms, no such extension of the algorithm was implemented.

## **Multiple Levels**

Finally, instead of inheriting the window size from the system surrounding it, the edges extraction algorithm could operate itself at multiple levels. This extension is closer to the biological approach, where parallel computation is used to distribute load from a single serial processor. When using



multiple levels, higher levels, which have larger window sizes, can reinforce the lower levels, which have smaller window sizes. By using this combined information, edge smoothing and even region extraction can be performed in parallel across an image. This process is also fully parallelizable. This approach then becomes similar to filtering, which is described in chapter 5.

## 7.4 Conclusions

This chapter presented a new algorithm for extracting the edges from a region. The algorithm assumes we have labelled the inside and the outside of the region whose boundary is sought. If these labels are not present, then a pixel-based metric such as color similarity must be used to determine if a pixel belongs to the region or not.

The algorithm presented uses a window-based approach to edge extraction which is both simple to implement, computationally cheap, and robust. Moreover, instead of complex mathematical operations, it uses simple averaging and estimation techniques which are feasible in a biological system. Finally, it can be implemented as a distributed algorithm.

The shortcoming of this algorithm is that it requires input parameters such as the window size, step size and the cosine cutoff for combining edges, that can cause changes in the output. These parameters could be combined within a single parameter determining the level of detail at which the edges should be extracted.

## Chapter 8

# Shape Representation and Region Matching

*Thou com'st in such a questionable shape, That I will speak to thee.*

William Shakespeare, Hamlet

The previous chapter described a way to extract boundary edges from an image region. This set of edges is the basis for the various shape descriptors we use. From the same set of boundary edges, different representations are constructed. Each emphasizes a different criterion one can use in comparing two shapes. Those criteria of comparison are area overlap, relative edge positions, orientation of edge normals, and the number and size of protrusions. These four ways of describing shape will be the topic of this chapter.

### 8.1 Literature on Shape Descriptors

Shape description is a fundamental problem faced in image retrieval, as well as vision and graphics. From each of these fields, we have chosen a representative example that influenced us in our approach to shape description.

### 8.1.1 Image Retrieval and Shape Description

Previous work on descriptors for image retrieval includes mostly global processing of pixel positions and wavelets. Shape is rarely discussed. In fact, shape is mostly unused in image retrieval systems.

The exception is probably BlobWorld [5], in which generalized ellipses are used to represent shape. This method can be thought of as a series of ellipses superimposed adding or subtracting detail to the shape. The first ellipse gives a rough estimate of the shape, mostly of its orientation. Subsequent ellipses add more detail.

Generalized ellipses are a very successful shape descriptor. They provide noise tolerance when few ellipses are used. Moreover, they provide incremental addition of detail, which can make comparison more efficient, by pruning bad matches early in the search tree. We hope to integrate in the future an algorithm based on generalized ellipses as one of our shape descriptors in Imagina. At this stage, we mostly experimented with four new shape descriptors.

### 8.1.2 Vision and Shape Description

Shape description has been largely discussed in the computer vision literature. We won't review here the various approaches here. We are mostly interested in representations that provide multiple resolutions. The most noteworthy example of such a representation is [38], that uses hyperbolic evolution of shape to create a hierarchy of shape features. The shape simplification that occurs is analogous to viewing an object from various distances. A follow-up paper [37] uses this idea for shape matching, and results in a language for describing shape. The base of the description closely matches the medial axis of a shape, the locus of all centers of spheres inscribed in a figure that touch exactly three points in the boundary.

### 8.1.3 Graphics and Shape Description

The computer graphics problem that inspired us the most for shape representation is that of morphing. Morphing is the procedure of transforming one shape into another in a continuous progression. If we have a way of assigning a cost to a particular morph between a source shape and a target shape, then we can use that metric to evaluate shape similarity. The best match between two shapes will be the one which has the least morphing cost.

The problem with most current morphing techniques is that the task of finding edges in the picture, and matching source and target shapes is left to the user. This is a reasonable thing to do

if the goal of the system is to create a good looking morph. Our goal however is to distance the user from such low-level decision on where edges arise and how they are transformed. We already have a way of extracting the edges. We now need a way to find how well the edges of one object match the edge of another.

Manolis Kamvysselis [18] proposed a novel technique for finding edge correspondences in morphing convex shapes. This technique first maps the two shapes into their Extended Circular Images (ECI), and then does the morphing in the ECI space. A one-to-one mapping exists between convex shapes and their ECI representations. Hence, every intermediate ECI obtained can be transformed into the corresponding shape. The ECI space seems like a good for morphing where such a metric can be found, and the results obtained in morphing shapes in the ECI space are promising.

The ECI representation of a shape maps every edge to a weight on the unit circle (see figure 8-1). The angle of the edge normal determines the position where the weight will be placed on the unit circle, and the length of the edge determines the weight that will be placed there. Morphing can then be done by simply transforming the weights of the source to the weights of the target. In ECI-morph, the transformation is simply an all-to-all mapping of weights based on an inverse exponential cost function taking into account both angle difference and weight difference.

What matters more to us is the overall cost of the morphing transformation, rather than the cost breakdown among individual edges. We are interested not in the way individual edges map to each other, or the particular matching algorithm found, but instead in finding a metric for an overall cost of transforming one shape into another. Such a morphing metric is the best candidate for a shape matching metric.

## 8.2 Shape Descriptors Used in Imagina

Since each representation is designed for comparison, the representation and the corresponding comparison method are best described jointly. This section will describe in detail each shape-based representation, its normalization metrics, and the comparison algorithm.

The different shape-based representations built on top of the boundary edges are:

- **Volume based representation:** Two regions are similar if they have a large number of common points when they are superimposed and adequately scaled.
- **Segment based representation:** Once aligned and scaled, two regions are similar if the

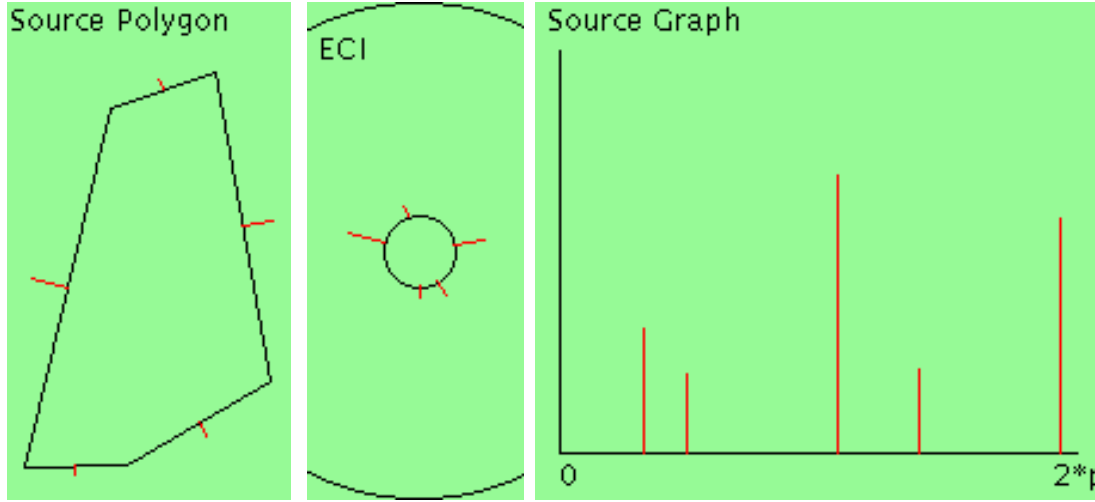


Figure 8-1: The Extended Circular Image (ECI) of a Convex Shape is constructed by mapping edges to weights on the unit circle. The left frame displays the original polygon along with the normals to the edges. The normal lengths are proportional to the lengths of the edges. The middle frame displays the weights on the unit sphere. The third frame displays a graph of the ECI weights obtained by unrolling the unit circle onto a flat axis.

segments making up the boundary of one are nearby those making up the boundary of the other, and vice-versa.

- **Angle based representation:** Once perimeters have been adjusted and the regions rotated, two regions are similar if for every angle between 0 and  $2\pi$ , the edges with normals facing towards that angle have similar sums of lengths.
- **Protrusion based representation:** Two regions are similar if they have similar protrusions located at equivalent points around the convex inner perimeter of the shape.

Each representation has different applications and strong points. This chapter will describe the representations developed, the scaling and alignment metrics used, and the matching algorithms developed.

In the remainder of this section, we present a few simple region descriptors, based on different features of geometric shape. We use the boundary edges extracted at the previous processing step as the basis for these descriptions.

## 8.3 Volume Based Representation and Comparison

The simplest representation is simply that of the set of points that belong to the inside of region. Comparison is done by superimposing two scaled and aligned regions, and comparing the number of pixels in common to the total number of pixels.

### 8.3.1 Approximating the Inside Points

From the set of edges, one can find all the points inside the region by using common computer graphics techniques. However, such techniques are expensive, and moreover, we already have the pixel information from which the edges were constructed in the first place.

We need however to approximate the set of inside points to make the expensive computation of comparing every pixel in a region more feasible in an interactive query environment. The way we do that is by undersampling the picture. Instead of considering every single pixel, one can consider one pixel out of 4, one out of 16, and so on, with similar results. In fact our assumption that the shapes are filled gives us an easy argument that such a measure can be adequate without a need to sample every point. Our preprocessing step of filling the shapes guarantees that this assumption holds.

### 8.3.2 Scaling and Alignment

The shapes are scaled so that they have the same areas. Since the area overlap is used as the metric of comparison, unequal surface areas could bias the results towards thinking that a small area entirely included in a larger one matches perfectly with it. Similarly, a large area would always have a relatively small area of overlap with a smaller region entirely included within it. These two biases do not balance out, since each of them saturates the results in the entirety of the small region, in the case of complete containment. Scaling the two regions so that they have the same area gives the desired size invariance of the comparison.

The shapes to be compared must also be aligned for area of overlap to be a significant metric. The two regions are aligned so that their centers of gravity coincide. An alternative would be to align the bounding boxes of the two regions. However, the bounding box is a metric that depends only on the extremities of the region, rather than the entire inside of the region. Therefore, if a silhouette of a person with both hands extended to the right is matched with that of a person with only one hand extended, the result is going to be very dissimilar. If however the center of mass is

used as an alignment position, these two shapes will be more rightfully judged more similar.

The shapes are not stretched, nor are they rotated. We believe that the aspect ratio of a shape is an important metric that should not be lost in scaling. Also, the angle based representation takes care of rotating the two shapes, so we did not want to take care of that concern here, especially since it would make a straightforward computation much more expensive.

### 8.3.3 Similarity Metric

Similarity can be measured in two ways. One way is to simply ignore the original color that was used in separating each shape, and return a value of 1 for every pixel in common, and a value of 0 for every pixel that is not in common. The sum is divided by the total number of pixels, and we have a similarity metric between 0 and 1. In this case, we are simply comparing areas of the two shapes.

Another way of comparison goes a step further by considering more degrees of membership to the region than 'inside' and 'outside'. For a region, the color similarity of every pixel to the average of the entire region in which it belongs can be used to determine a membership value that can be anywhere between 0 and 1, rather than just 0 or 1. The use of color here is different than the one in the color-based comparison. Color is not compared between the two regions. Color is instead used within each region to determine the degree of membership of a particular pixel to the overall regions that it represents.

If this similarity value between 0 and 1 is considered to be the thickness of the shape defined by each region, then the overlap between the two regions becomes the volume that the two aligned shapes have in common. It is therefore not a planar comparison of surface areas that this algorithm performs, but instead a comparison of volumes in two and a half dimensions, where the thickness is the similarity of the shapes, between 0 and 1.

### 8.3.4 Results

Figure 8-2 shows an example of a volume-based comparison on two images of the St. Louis Arch. If one considers the color histograms of the two images, they will appear very dissimilar, as is shown in figure 8-3. Having a shape-based comparison allows an robustness to lighting conditions when searching for a particular image.

Many current image retrieval systems that rely heavily upon color information will be less likely

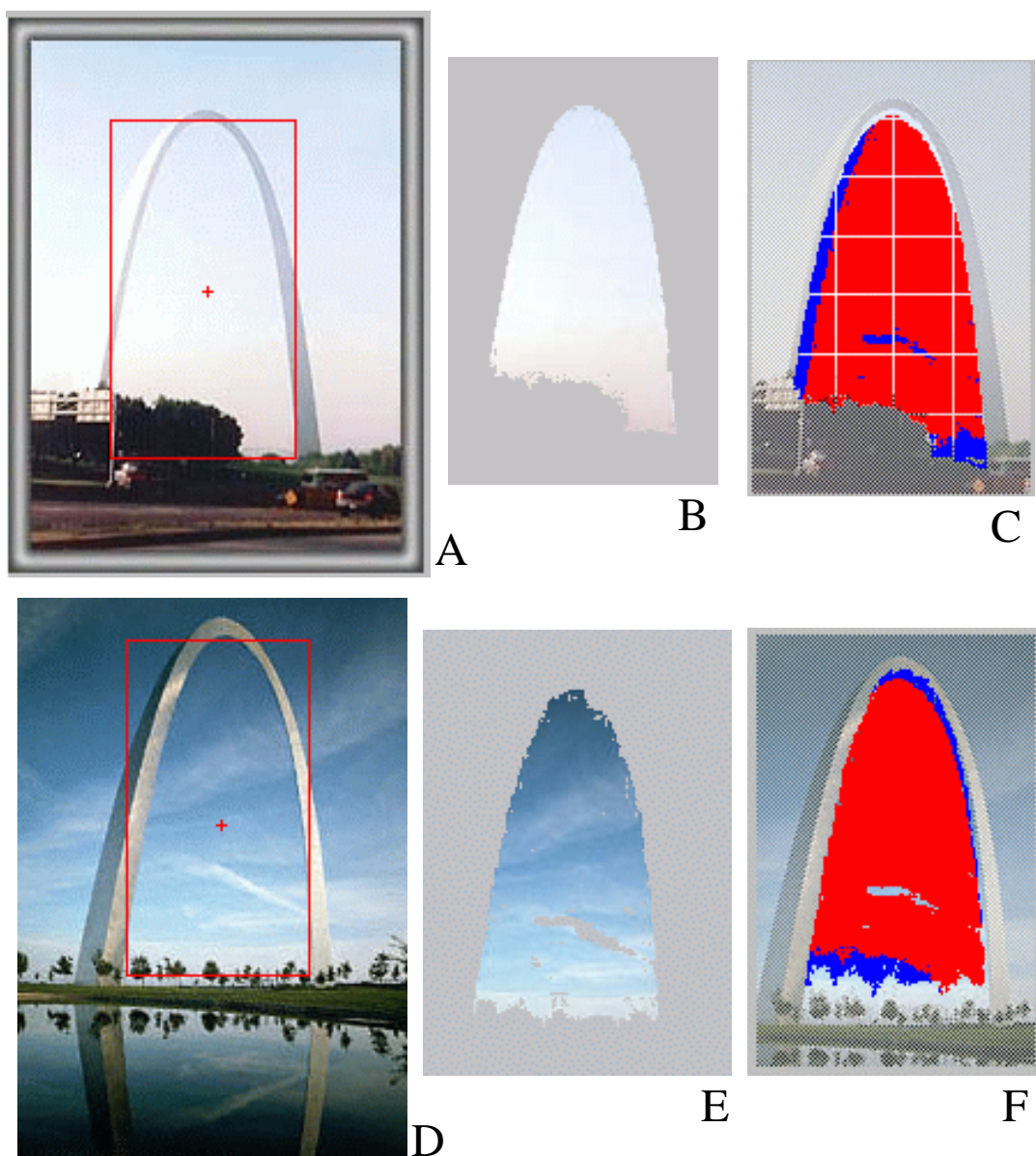


Figure 8-2: The Volume-Based Comparison. **A** shows the original image, along with the center of mass of the region's pixels and the bounding box around the selected region. **B** shows the pixels belonging to the region. **C** shows in blue the pixels in B that do not also belong to E and in red the pixels that B and E have in common. Similarly, **F** shows in blue the pixels that belong to E but not in B, and in red the pixels belonging to both B and E. The similarity metric returned is the number of pixels in common (red) over the total number of pixels (blue and red). The similarity metric returned for this match is 0.87.



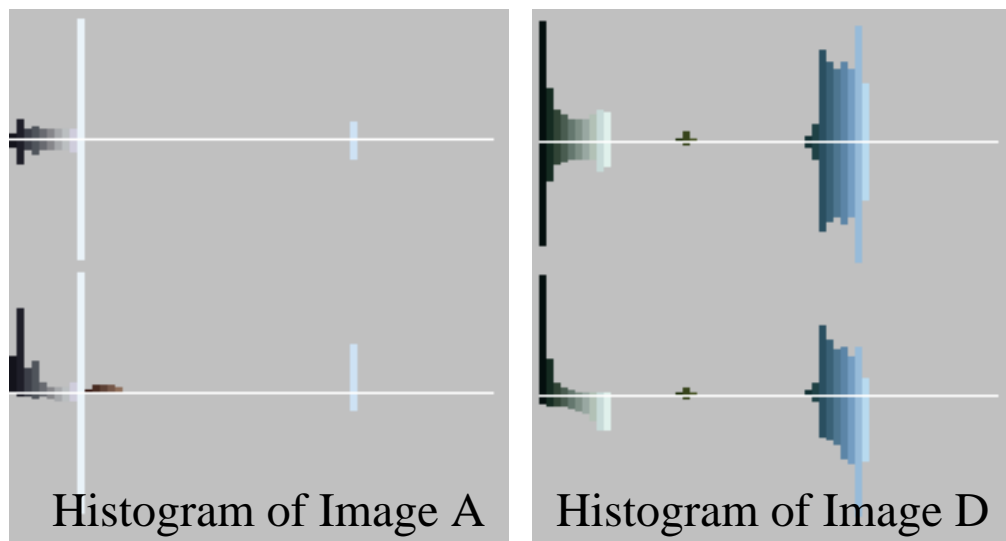


Figure 8-3: The color histograms of Images A and D have a similarity of only 0.19. The horizontal axis represents different color buckets, and the four vertical axes represent from top to bottom number of points in a bucket, standard deviation of Hue, standard deviation of Saturation, standard deviation of Value, in the HSV color space. Picture D spans the entire blue color bucket, while picture A contains primarily one spike in a single tone of blue. Moreover, the shape of the distribution of grey in the two images doesn't match.

to find the match between these two images. This is demonstrated in the results returned by the Altavista Image Finder when picture A is used as the query to a similarity search. The best matches returned are displayed in figure C-3 in appendix C. The Virage engine for matching images does not seem to give any importance to shape in this example.

In Imagina, the shape of the region under the two arches is a salient feature to match, since it's large, uniform, and centered. It will be segmented as a single region, since the color under the arch is uniform. Its shape will then be matched to the shape of the corresponding region in the target picture, and a high similarity will be returned.

### 8.3.5 Shortcomings and Strong Points of a Volume Based Representation

This representation is best fit for matching regions that have roughly the same overall shape but whose boundary details may diverge. The strong point of it is a tolerance to noise created by holes in the image. Geometric representations will consider a shape with a hole as a topologically different object from the filled shape. Similarly, representations based on edges will tend to give more importance to high frequency changes in the region boundary, while a volume-based representation

will be unperturbed.

The main disadvantage of this method is that it breaks when the alignment between two shapes is not adequate. This situation arises if the segmentation for one image assigns a larger portion of the image to a region on one side, and segmentation for the other image assigned a larger portion of the image on the other side. In that case, the two images will be misaligning, and the two shapes won't match. This situation arises too often for thin shapes, since their overlap can be reduced drastically with small misalignments.

An example where this representation breaks is the matching of Marilyn to a drawing that will be our criterion for the rest of the representations.



Figure 8-4: The Volume-Based Comparison breaks for figures with many details. For example, on the image on the right, even though the dress of Marilyn has the same overall shape, it is misaligned due to noise in the segmentation, and this causes the area of the hands to be miscounted.

## 8.4 Segment-Based Representation

The segments-based representation comes as an answer to the fact that the volume based representation breaks for thin images. In the example of Marilyn, when her hands are misaligned, the match suffers heavily. Finding a better alignment for the volume-based representation might be

an answer, but again, small differences in angle that do not matter to the user will count heavily against a volume-based match.

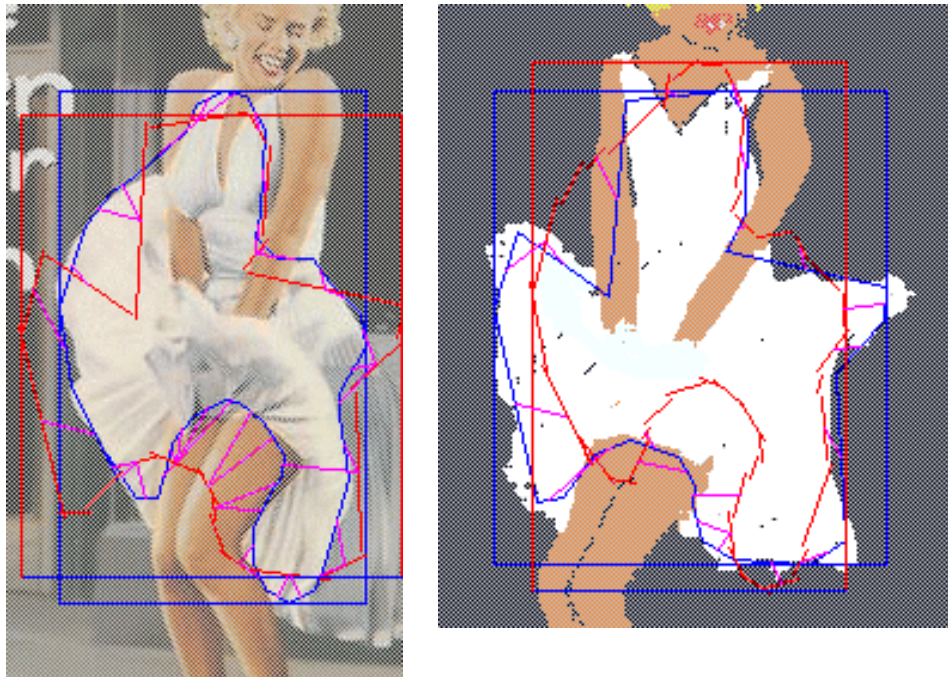


Figure 8-5: The Segments-Based Comparison. On the left, every edge on the original image is matched with its nearest edge in the query. Segments are drawn between the midpoints of the matched edges as a visual feedback. The sum of the lowest costs for every edge becomes the cost of mapping database edges to query edges. On the right, each query edge is matched with the best database edge. Similarly the cost of mapping the query to the database is computed. Together these two costs become the cost of transforming one shape to the other. The value obtained in this match is 0.94. Of course, this value alone is meaningless, since it will be normalized according to the distribution of all segment based matches, when the matches from different representations are combined.

The metric of similarity in the segments-based representation takes into account the proximity of the figure's boundary edges. Proximity is calculated on an edge by edge basis. The metric used is comparing two edges is based on coordinate proximity of the segments midpoints, angle similarity between the two normals of the edges considered, and the length ratio between the edges. These three metrics of every edge are combined in an overall cost of matching one edge with another. The cost and the similarity are complementary of each other, and are both between 0 and 1. Their sum is always 1.

The aspect ratio of the regions is preserved, but the regions are scaled and aligned with each

other. We believe aspect ratio is important in matching images, and the users have a good idea of the relative dimensions of a shape they are looking for. Moreover, the aspect ratio is often invariant between different pictures, although the pictures can be found in all sizes and alignments within a picture.

The two representations are scaled so that their bounding boxes have equal diagonals. The bounding box is used since this representation is based on boundary edges, hence if the boundary matches, the bounding boxes will be similar. The diagonal of the bounding boxes is used instead of the area, because it gives a more uniform rate of change with changing dimensions of the bounding box. That is, when one dimension becomes dominant, the diagonal does not vary geometrically, while the surface area would.

Finally, the two regions are aligned so that the centers of their bounding boxes coincide. This metric is used instead of the center of mass which would not be meaningful in a representation based on boundary edges rather than volume. Moreover, by aligning the centers of the bounding boxes, this brings the edges closest to each other overall.

## 8.5 Angle Based Representation and the Extended Gaussian Image

Based on the work of Kamnitsas [?] described in section ?? we construct an angle-based representation for shapes that relies on the Extended Circular Image (ECI).

The difference from the original ECI representation lies in the creation of angle buckets that contain more than one angle value. By choosing the number of buckets, one can determine how precise the ECI representation will be made. This provides even at this representation level another metric for noise tolerance or attention to detail. Our experience shows that 20 buckets are adequate for matching most shapes. Fewer buckets give too coarse descriptions and more buckets tend to separate almost similar shapes.

The Angle-Based representation creates a number of buckets that contain different partitions of the angle space and goes around the edges, consecutively adding to each bucket the length of each edge belonging to the bucket.

Comparison is done at the bucket level, by simply measuring angle differences between nearby buckets and length added within each bucket. Comparison between angleBuckets with different number of buckets is disallowed, but can easily be extended by reprojecting and recalculating the

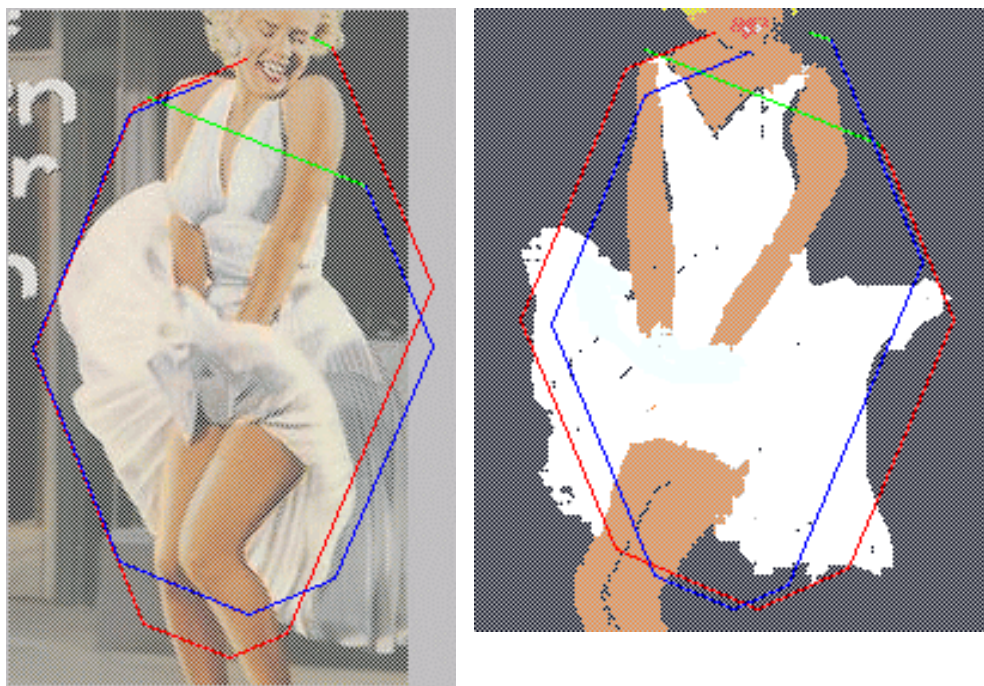


Figure 8-6: The Angle-Based Comparison for Marilyn with only 8 buckets. The two representations are rotated for achieving an optimal match.

angles that the corresponding buckets would give.

The debug information shows the buckets painted as edges with the corresponding length and the middle angle of the bucket, centered at the center of the figure.

The strong points of this representation is that it is rotation invariant, and tolerant to small errors in angle, when the overall orientation and derivatives match. In that case it is similar to the generalized ellipses described in [5].

## 8.6 Shape Descriptor based on Protrusions

From the cognitive science community, we borrow the idea that protrusions is what humans base their recognition upon. The shape of crevasses in the region boundary does not matter, but instead the protrusions that bulge out of an image is what humans have learned to recognize.

Based on this assumption, we constructed a representation that uses mainly protrusions in the matching. Protrusions are extracted from the edge-based representation of the boundary, by considering adjacent edges two by two, and using the angle between them to label each edge as a protrusion edge or an inner-perimeter edge.

Once the set of protrusions for each image has been extracted, the protrusions are matched in order based on their position on the inner perimeter, the outer perimeter of each protrusion, the length of the base, and the orientation of the protrusion's center of mass with respect to the base.

Scaling is done by aligning two separate but not independent metrics, the inner and the outer parameters. The inner parameter is formed by the non-protrusion edges, along with the bases of protrusions. The outer perimeter is the sum of the lengths of protrusion edges.

## 8.7 A metric for combining comparisons from individual methods

Instead of returning an absolute proximity metric, we return a relative one. That is, when two different representations are used, say based on color and based on shape, we have to have some way of comparing the outputs of the corresponding comparison functions in the two different domains.

In each domain, the comparison function returns a value between 0 and 1, where identical representations that are 100% similar have a similarity value of 1 and dissimilar images have a similarity value of 0. However, this scheme breaks down when say, every image in a particular representation



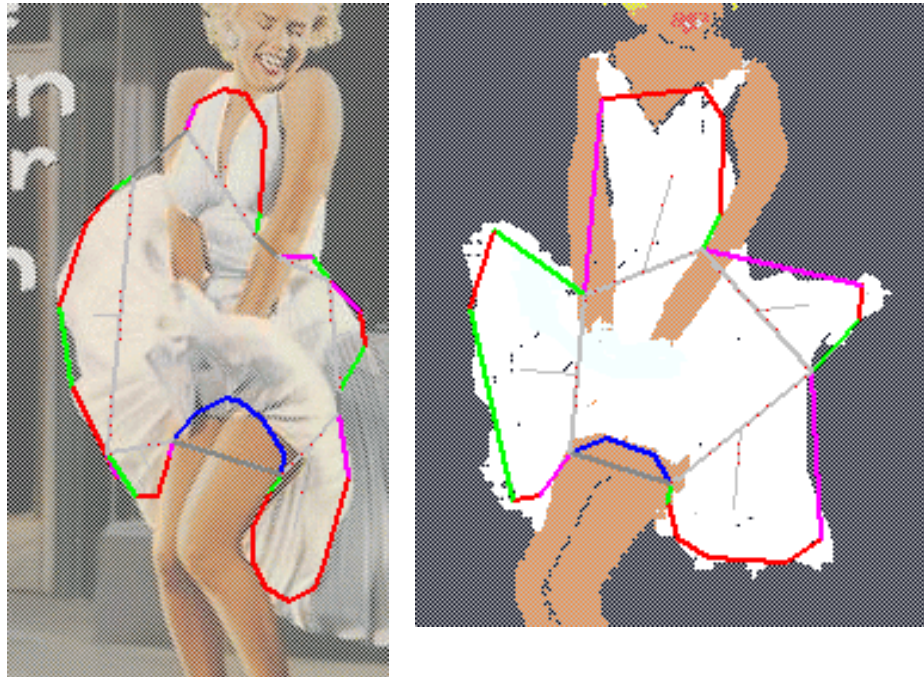


Figure 8-7: The Protrusion-Based Comparison. In grey is painted the inner perimeter of the shape, ignoring protrusions and cavities (drawn in blue). Going counter-clockwise, the first edge of a protrusion is drawn in green, the last one in magenta, and the other ones in red. Two shapes are similar when they have the same number of protrusions at approximately the same locations on the inner perimeter and the same orientations relative to the base.

has the same representation. Thus if a representation is broken it might advertise a similarity of 1 with every other image, and might bias the search to use that particular representation.

Therefore, we will use a relative metric based on these absolute similarity values. This metric measures the similarity of the query to a particular image as compared to its dissimilarity to the rest of the images. Therefore, for a particular representation, we will return as the proximity metric of the best image the ratio between the similarity value of the best image and the average similarity value.

## **8.8 Conclusion**

This chapter presented a combination of different approaches for describing shapes, each having specific strong points and being tailored to specific applications. All of them have weak points, but when one fails usually the others take over, thus providing a graceful degradation of the shape description.



## Chapter 9

# Image Retrieval and Matching

*Though we travel the world over to find the beautiful, we must carry it with us or we find it not.*

Ralph Waldo Emerson

Image retrieval and matching brings together the work of the previous chapters into the system of Imagina. All of the methods of region matching which are used in Imagina have already been presented. The next step is the combination of the representations into a unified measure of match between regions. Image matching based upon this, and a simple configuration measure, is then straightforward.

Once the similarity between regions can be measured, the similarity between images can be computed. This measurement then allows for the comparison of similarity between a query and the images stored in the database, yielding a content-based image retrieval system.

The problem of image retrieval is not as straightforward as this might imply. Image retrieval is a hard problem because the problem itself is ill-defined. Given a query image and several database images side by side, few people would be able to agree on which of the images provides the best match. This, however, is exactly what an image retrieval system is expected to do.

## 9.1 Challenge of Classification

Classification, whether in terms of image segmentation or shape matching, is an ill-defined problem. Without a bias for a category of solutions, a useful answer can never be achieved. How else can the “right” answer be identified? Although intuitively the “simplest” answer matching the desired criteria is the right one, this is not always as straightforward as it seems from our evolutionarily advanced vantagepoint. Computer systems, lacking such a concept, need a more stringent definition.

The following example illustrates this. The example is borrowed from [26, pages 21-22]. Take a minute to look at the training set on the left, determine the classification rule which is being sampled, and consider possible test set item labellings.

	Training Set								Test Set							
Attribute 1	1	1	0	1	0	0	1	0	0	0	0	1	1	1	0	1
2	1	1	1	0	0	0	0	1	0	0	1	0	1	1	1	0
3	1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1
4	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1
Category	A	A	A	A	B	B	B	B	?	?	?	?	?	?	?	?

Table 9.1: Pattern classification task example.

In Table 9.1, several rules may be valid. For example, the label ‘A’ could represent the NXOR of attributes 3 and 4, while the label ‘B’ could represent the XOR of attributes 3 and 4. Alternately, A could be the AND of attributes 1 and 2 ORed with the NXOR of attributes 2 and 3. A variety of other solutions come to mind. The basic difficulty lies in that, since the entire space of solutions can never be seen in its whole as examples, the classification rules must make assumptions about the space they are operating in. Otherwise, the rules would never be able to classify unseen examples.

This problem only becomes more difficult when noise is introduced into the system. Then, errors would occur even if the ideal classifier were to be known. In case this problem seems too simple, consider this: the solution which seems simplest based on the number of attribute combinations required, the XOR of attributes 3 and 4, yields a homogenous pattern labelling for the test set. Given that the training set had two equally-represented sets, would it not make sense that a random test set would contain roughly equal numbers of examples from each category? The correct choice of classification rule is, after all, dependent on the task that is being solved.

## 9.2 Combining the Representations

In performing the image retrieval version of classification, unrelated measures provided by different representations need to be combined. This is done through the application of the concept of how to combine unrelated measures presented in section 3.6. This idea was applied in Imagina both for the region and the image search implementations.

For region searching, the similarity measures computed by the comparison of the different representations which are abstracted from each region must be combined. Based on this idea, the measures are first normalized before being summed into an overall match score. In order to be able to do this, the distribution of every measure has to be normalized. Therefore, assuming a normal distribution for each similarity measure computation, the mean and the standard deviations of each distribution must be computed.

In a large database the distributions can be approximated through random sampling. In a small database, however, the distribution can best be approximated by looking at all of the sample points available. Therefore, regions are matched sequentially against all other regions in the database.

The distribution of results of one match may not be a good predictor for the distribution of results for another match. Therefore, the distributions of the similarity values returned by each representation are recomputed each time a region gets matched.

Every computed similarity is then normalized as in:

$$\frac{value - mean}{standard deviation} \tag{9.1}$$

This results in a distribution with a mean of 0 and a standard deviation of 1. Thus, all of the representation similarity evaluations end up having the same distribution. The normalized similarity measures for each representation are then combined into an overall similarity measure, resulting in a measure of the similarity between two regions. The regions are then returned to the user as ordered by this combined measure.

## 9.3 Dealing with Multiple Regions within an Image

Once the region similarity can be computed, the similarity between regions can be used to perform image matching. However, the configuration is also a feature which is important to users of an image retrieval system. Therefore, a similarity measure of two different configurations of regions

must be computed.

Configuration can be encoded and computed in a variety of ways. It can be used either as a first step in indexing a probe image, or as a last step. If it is used as a first step, then a graph-matching algorithm must be performed. Region comparison, however, is simplified, because the space of possible regions is greatly diminished. On the other hand, if it is performed as a last step, it could optionally be skipped, unless the request retrieves a large number of matches.

One configuration encoding uses not only relative position and size, but also some shading information. Sinha explores such a representation in chapter 7 of his PhD thesis.[39] His representation uses an encoding of the relative lightness of different equiluminant patches of an object, for example, of a human face. The shape of the patches is left to be rectangular, for ease of computation and matching, and the images are compared in black and white. Although color could potentially be used, a single-valued relationship between colors could not be extracted. By looking at colors only in terms of their intensity, however, such an encoding can be achieved.

Such an encoding would allow the matching of whole images before the matching of individual regions is even attempted. Region matches would simply provide an improvements to the estimation of an image match. However, queries would have to be fairly complete in order to allow this matching method to work. If too few regions were supplied, or if the relative shading were wrong, the indexing would fail. Thus, this method would work best for indexing based upon real image probes. For Imagina, where a user sketch input is the primary indexing method, this is not a first choice.

## 9.4 Image Matching

The same idea which was applied to region matching is applied to image matching. In this case, the distribution of similarity has to be computed for each region in the query image. That is, each query image region has to be compared against all regions in the database, and the resulting distribution has to be normalized individually for each region. Again, for an extended database, sampling could be used to approximate this distribution.

The only remaining issue is the computations of configuration similarity. This issue is resolved by placing this computation at the point where two sets of regions have been considered as a match. Because the mapping between the query and database region subsets, of the query and one of the database images, respectively, is known, the configurations can be compared directly.

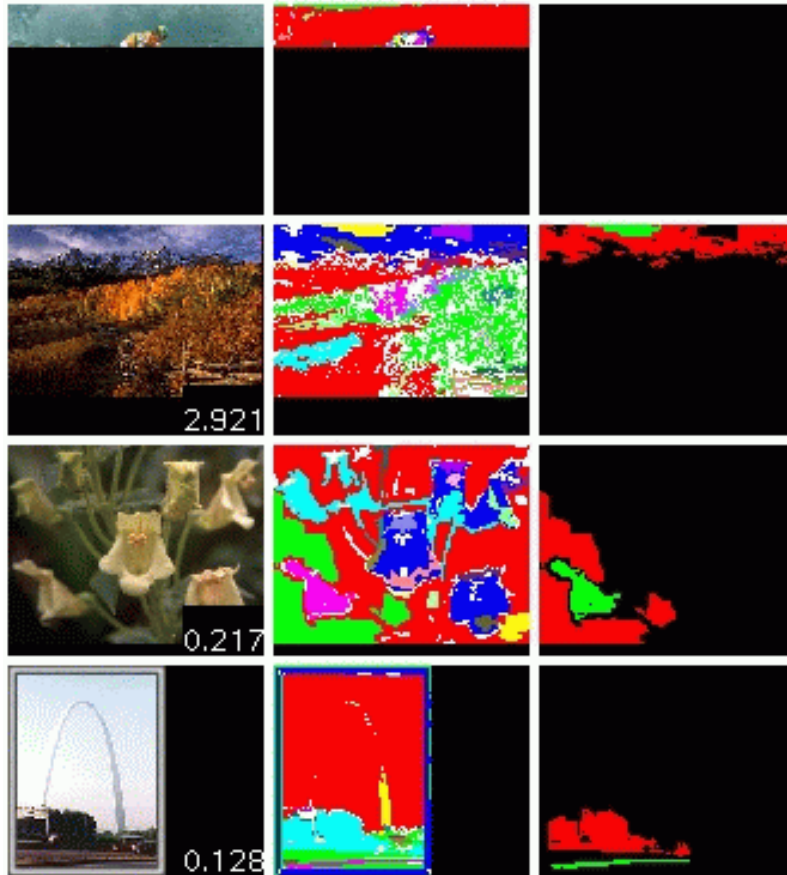


Figure 9-1: The first successful image match performed by Imagina. The query image is displayed on the first row, and the matched images are listed on the subsequent rows. The first column is an image thumbnail, the second column is the image mask, which shows the different automatically segmented regions of each image, and the third column shows the mapping of the query image regions to the matched image regions by the use of color coding. Thus, the region which is labelled red in the query image matches the regions which are labelled red in the third column of every image match.

### 9.4.1 Comparing Configurations Given a Region Mapping

When a configuration is evaluated, a mapping from the query image regions to the comparison image regions is given. To evaluate the configuration similarity of two sets of regions, A, B, C and A', B', C', we use three kinds of similarities: angle similarity, distance similarity, and relative areas similarity.

When comparing the configuration of the regions A and B to the configuration of regions A' and B', we consider the center of mass of each region to be representative of the position of the region and the number of pixels belonging to the region representative of the volume. We therefore evaluate the cost of transforming the pair A, B to the pair A', B'. The cost is made up of three components:

- The angle between AB and A'B' is normalized with respect to  $\pi$ , the maximum possible angle difference.
- The relative areas are compared. The ratio of B to A is compared with the ratio of B' to A'. Again a ratio similarity between 0 and 1 is returned.
- Finally the distance difference between AB and A'B' is computed, normalized relative to the square root of the areas.

The resulting similarities are combined in giving the similarity of pair AB with pair A'B'. The total configuration cost of region A is the sum of the costs of all pairs AX and A'X', for every other mapping XX'. This gives a configuration cost for A, to be combined with the region similarity cost.

This results in a value from 0 to 1. This value is then used as a multiplier to the region match, for the given region mapping being evaluated. These weighted, previously normalized similarity values are then summed up to give an overall image similarity value for the given attempted mapping. Thus, the similarity of an image to another image is in terms of the configuration which was attempted. The user can opt to see one or several of the top configuration matches which the query image has made with every database image.

### 9.4.2 Matching Issues

The only problem with this approach is that the search space - all possible region mappings between one image and another - have to be considered in turn. This search is of the order of  $\frac{\max(m,n)!}{\min(m,n)!}$ ,

where  $m$  and  $n$  denote the number of regions in the query and the database image, respectively. Since the number of regions in database images vary, the worst case order of growth is given by the largest number of regions present. However, the order of growth by the number of images in the database is only linear. Therefore, an improvement in the local image matching is the key to improving matching performance.

Although the growth rate of the function might seem large, for a small database a system programmed in Java the limitation of memory size is reached much more quickly. Extensions to the database design are discussed in chapter 10. Results of several runs of the system can be found in the appendices. Appendix D shows whole image matches, and appendix E shows region matching.

# Chapter 10

## Extensions

*You see things as they are and ask, ‘Why?’ I dream things as they never were and ask, ‘Why not?’*

George Bernard Shaw

Several extensions to the existing Imagina system are possible. These can be grouped into several categories:

- **Preprocessing Images** A very general improvement would be to increase the image fidelity for the images inserted into Imagina system by applying a pre-processing step. This can be done to improve image indexing performance in almost any image retrieval system.
- **Creating an More Extensible Database** A database which is more computationally efficient and can therefore handle larger numbers of images gracefully would allow for the further testing of our concepts. We present an approach to a creating such a database in this chapter.
- **Learning from User Feedback** User feedback, other than the modification of a few search weights, is currently not used. By implementing a process which observes the user’s interaction with the system, search results could be improved.



- **Additional Algorithmic Modules** Additional matching modules can be added to the system to improve search performance. This is easily done since the modules themselves are interchangeable.

## 10.1 Preprocessing: Improving Image Fidelity

A difficulty with image indexing and segmentation is that not all images are taken in the same lighting. Some images are very dark, others are very bright, and others are shaded unevenly by the different colors of light. This difficulty is often not obvious to the human observer of a scene being photographed because the human visual system automatically compresses the dynamic range of lighting and effects color constancy. Thus, objects look the same to humans under a variety of lighting conditions, while to a camera they can be quite different. Professional photographers are well-accustomed to this, and use light-measuring tools to aid in taking pictures which look realistic to end users.

An approach to bypassing this problem was suggested by Edwin Land in terms of the center-surround retinex concept, introduced in [23]. This approach has been taken up and implemented by Jobson, Rahman et al.[16][17] In their system, a center-surround region is computed at every image location, with the filter consisting of a strong weighting for the current image pixel, and a small negative weighting over a large area of surrounding pixels. They developed multi-scale version of this algorithm, which provides a good color image enhancement.

The central difficulty with applying this system is twofold. First, several constants need to be determined empirically for good performance. Second, this technique is being patented, and therefore not likely to be freely available. The appeal of this system, however, is the claim that a wide variety of input images, from indoor scenes to space images, can be processed automatically equally well.

The retinex approach, or another color image enhancement technique, might be a useful preprocessing step for any image indexing system. This is especially true about systems which are based upon color histograms. These are usually tested on a select data set, such as a Corel image collection, which do not exemplify users' image collections. For Imagina, the improvement would lie in that a larger variety of images would be segmentable. Currently, images which have little variety in the tones of color, either being too dark or overly bright, are difficult or impossible to process in a manner which is useful to the end user's search process.

## 10.2 Database: Principles of a Large Database Design

A large database needs to handle many items while at the same time performing computationally nontrivial matching. Although the individual modules perform matching at query time which is on the order of several thousand computations, combined across different modules and the entire search space this can become overwhelming. Instead, a system based upon a hierarchical definition of the database into prototypes and represented subsets becomes useful.

In this, and throughout the current implementation, the guiding principle in the Imagina database is that new inputs into the database are classified based on the inputs already present in the database. The database is expected to be roughly the same regardless of the order of insertion, however. When a new item is inserted, it is segmented into regions based on different algorithms. These regions are then defined by separate algorithms, the output of each being placed in a different *description map*, as shown in Figure 10-1.

We define a description map to be a complete indexing of the regions in the database, as described by the application of a single series of algorithms from the starting image. Therefore, the database is comprised of several description maps, with each region which was segmented from the input image being represented in each of them individually. Matching is performed by finding the nearest neighbors in each map, and then correlating neighbors across maps.

As aforementioned, as the complexity of the database increases, prototypes can be used to simplify the matching process. A Prototype can be abstracted at insertion time if a given subset of abstracted images within a single description map are sufficiently similar. When matching is performed subsequently the search can be pruned at the prototype if the similarity of the prototype to the search target is sufficiently dissimilar, saving computation that would be spent on equivalent matching being performed at a lower level of the hierarchy in a given description map. The idea of prototypes can also be applied hierarchically, to allow for the matching computation to grow in proportion to the logarithm of the number of images in the database.

## 10.3 Database: Representation of Images

Images are usually stored in databases in terms of a total ordering of all of the images within the database. That is, each image  $I$  is mapped to some representation  $R(I)$ . The function  $R$  is more or less complicated, but yields usually a unique representation for each image. The representation

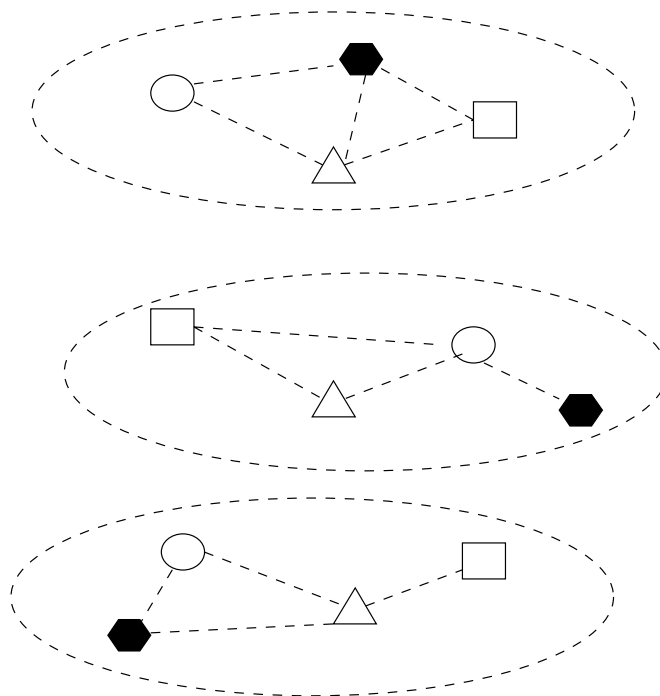


Figure 10-1: A graphical display of four regions as they map into three distinct description maps. Each individual region is denoted by a different shape, and a dashed line is drawn from it to its nearest neighbors within that description map. A new region, displayed here as the filled-in hexagon, can have different nearest neighbors in each abstraction map.

$R(I)$  is added to the database  $D$ .

The alternative is to use a graph abstraction to store the images. Instead of considering every image during a search, only a set of nodes of the graph are considered.

In this case, the new image  $I$  is added to the database relative to the other things that the system has discovered so far. The representation of the image is thus dependent on the previous images viewed. We compute  $R(I, D)$ . This representation is a function of both the image and the previous images.

Moreover, this new image changes the connections and relations of many other images in the database. Hence the database itself is transform in a more complex way than simply adding to the database the picture  $I$  indexed by the representation  $R$ .

## 10.4 Database: Structuring a Large Database

The database is usually a set of images, somehow interconnected. In a system like Altavista, a lot of images are stored along with their pre-computed relationships, in order to achieve a quick response time. In other image databases, different indices might be used to cluster the data for a faster, retrieval using those indices.

Our database does not only contain images. It also contains abstractions of images, at different levels. Different abstractions from the same image will yield a mapping to different such abstracted images. The representation of the new image then becomes the interconnections between these abstractions. These abstractions define the image. Because of this, the problem of indexing becomes the problem of graph matching, where the nearest neighbors of every abstraction of a given object have to be combined into an ordered similarity list.

## 10.5 Database: Abstraction and Specification Functions

The different filters, segmentations, and color-based matchings which are used for image indexing can be thought of as different functions, each of which takes a particular type of input, and produces a particular type of output. This description is very generic, because each function performs an action on a potentially different space. Although color and edge determination are both performed on an input image, their outputs can be very different in quality.

### 10.5.1 Abstraction Functions

We can think of the different ways of abstracting an image into something as applying different abstraction functions  $A_1...A_n$  to the same image  $I$ , and obtaining different abstracted versions of the same image  $A_1(I)...A_n(I)$ .

### 10.5.2 Different Levels of Abstraction

To have different levels of abstraction, there are two approaches.

- Make the output  $A_i(I)$  of an image  $I$  belong to the same domain  $D$  as the image  $I$ . Thus, the same abstraction functions can be applied recursively, yielding  $n^k$  abstractions for  $k$  recursive applications of  $n$  abstraction functions.
- Define new abstraction functions for every level. If we want five levels of abstraction, we can construct four abstraction classes  $A, B, C, D$ . The range of  $A$ ,  $D_A$ , is the domain of  $B$ , and so on. We can thus obtain  $n_A * n_B * n_C * n_D$  possible abstractions where  $n_L$  is the number of abstraction functions defined in level  $L$ .

### 10.5.3 Specification functions

The specification function takes an abstract idea and produces a possible incarnation of this idea. It is basically the inverse process of abstracting an image to obtain an idea. If from an image one can obtain a set of possible abstractions, then each of the abstractions can be made more specific as that image. However, it could also be made more specific as a different image, which however must be abstractable to this abstraction.

Defining specification functions is like defining the inverse operations of the abstraction functions. We can either require that each abstraction function be reversible, or we can simply define a set of specification functions  $S_1...S_m$  more or less independent of  $A_1...A_n$  with the single requirement that the domain of all  $A_i$  is the range of all  $S_j$  and vice-versa. That is we can simply require that the domains coincide going up the abstraction tree and down the specification tree. This is definitely easier to implement, and an exact match at any level is not obligatory since recovery of an image is never perfect, and the space of possible images and abstractions has no hope of being spanned.

### 10.5.4 Coding Abstraction functions

Let's assume for the moment that

1. The abstraction and specification functions are independent, That is one doesn't have to make abstraction functions reversible.
2. There are as many domains as classes of abstraction functions. That is, the domains of  $I$  and  $A_j(I)$  need not be the same. This results in a finite number of possible abstraction levels.

Both assumptions can be justified biologically from a cognitive science point of view. First of all, only a certain number of neurons are traversed in recognizing an image. Only a fixed number of neuron stages are involved, and we can assume that each stage is specialized in receiving the output of the previous stage. No circularity has been observed in the process of recognition, which would involve the domain and range of a particular function (possibly nested) would have to be the same. That is  $A_i(B_j(C_k(I)))$  does not have to be usable as the input to  $A_l$  for example.

Moreover, both assumptions make coding much easier. We can now construct different abstraction functions that extract different pieces of information from the image. The information extracted at each stage is used in the next stage in making its operations. To make it possible to have later stages use information from the previous stages, we simply augment the input  $I$  of a stage with the output of the function at that stage, and make the pair  $I, A(I)$  into the output available to the next stage.

The next stage for example could operate on either the output of the previous stage (generating  $\{I, A(I), B(A(I))\}$ ) or on the original image (generating  $\{I, A(I), B(I)\}$ ). This makes it possible to represent any non-cyclic graph as a tree.

However, there is a restriction is that at every level, the abstraction function applied must have an output which can be consistently interpreted. That is  $A_1$  cannot be computing the average green and  $A_2$  the average red in the image, otherwise the comparison function would consider a forest and a volcano as similar in color.

## 10.6 Learning: Using Feedback to Improve Search Performance

A longer-term goal of this project is to have a training phase, during which users will be able to search images in the database using a sketch of the desired image as the query. The system can then use the input sketches, along with the corresponding simplified and original images, to improve

the internal representation by adjusting relevance parameters on matching criteria. This feedback should yield a system that learns how to abstract images more accurately as the user is formulating queries and getting results, and hopefully improve the searching capabilities as time goes by.

### 10.6.1 Re-Weighting Functions

The specification and abstraction functions can be re-weighted based on the user's requests. By seeing which function was the most accurate predictor for the user's request in the previous query, its weight can be updated for the subsequent query.

### 10.6.2 Speed up questions

To speed up computation at the time of matching, the functions themselves can be precomputed on the images, and only put together at the time of matching to reconstruct the outputs. That is, we precompute  $A_i(I)$  and we construct  $\{I, A_i(I)\}$  only when needed, depending on the type of representation and the levels of abstraction.

## 10.7 Module Extensions: Parallel Implementations

Biological systems use very slow wetware to compute information which even modern-day computers performing hundreds of millions of computations a second cannot keep up with. Their advantage comes from using a large number of slow processing devices in parallel rather than using a single fast serial computation.

There are two reasons to follow up on this lead. First, even with ever-increasing computational speeds, serial silicon-based computation is reaching its limits. Only by using computing devices in parallel will computation be able to take the next leap. Second, we already know that a system is capable of performing the image indexing task which we are trying to set up.

We also know that the brain system is massively parallel in its computational approach. Therefore, algorithms which are readily parallelizable are both more likely to perform the desired computations, and more readily speeded up through parallel computational architectures if they are to be applied. Thus, one of the goals of the Imagina system is to develop or use algorithms which are easily computed in parallel across an input. Although this is a goal, at present a tradeoff has to be made towards a more computable and serialized implementation.

## 10.8 Additional Modules: Texture and Shadows

To manage the extensive task set out for this thesis, an incremental approach is adopted. To be able to handle the basic task of segmentation, color suffices in many cases. It was therefore implemented as a first step towards the implementation of segmentation algorithms. These, in turn, led to shape-based descriptors being defined and created.

One of the goals of Imagina is to be able to handle images of objects in uncontrolled situations. This requires that segmentation algorithms to be very robust, and use as much information as possible from the image. Ideally, color, texture, shape, and shading could be used. Unfortunately, these features require progressively more effort to implement and are less obvious to design. At this point, only color and shape are implemented into the modules used by Imagina. By taking advantage of these other sources of information in every image the matching performance could be improved.



# Chapter 11

## Conclusion

*Attitudes are much more important than aptitudes.*

Alexander Lockhart

### 11.1 Contributions

We feel Imagina has contributed to research in content-based image retrieval by suggesting novel approaches in several aspects of the problem.

- ★ Sketch Based Drawing
- ★ Global Segmentation based on color
- ★ Matching based on region shape
- ★ Multiple representations at multiple levels of detail
- ★ Explicitness of Representation

#### 11.1.1 Sketch Based Drawing

What sets Imagina apart from most current content-based image retrieval work is the sketch interface for formulating queries. Computation on the query is cheap, and can be easily compared

with precomputed representations of the database images. Other systems can only query in terms of precomputed images within the database.

The use of a sketch as an interface is leaving to the user the freedom of a greater versatility in expressing queries. We hope that this will make the user of image retrieval by content as natural as text retrieval has become, by making the medium of the search match the medium of the data. We hope that future image retrieval research will move towards systems that allow greater user input flexibility.

### **11.1.2 Global Segmentation based on Color**

This thesis presented a new algorithm for segmenting images based on color. This algorithm makes use of the different scales in an image to combine regions at different levels. The result is a robust algorithm to segment images based on color at the Hue-Saturation-Value color space. Also, the description of the HSV color space provided in this thesis is to our knowledge one of the few in the literature that cover all details correctly.

### **11.1.3 Matching based on region shape**

The use of shape has rarely been addressed in image retrieval systems. Imagina takes a new approach to shape by combining different representations each emphasizing a different feature of the region's contour. Imagina has presented a new simple and robust edge extraction algorithm for constructing the various shape descriptions. Imagina has contributed in presenting four different representations that complement each other in matching shapes and can be used in parallel. This successfully achieves the graceful degradation principle of biological systems.

### **11.1.4 Multiple Levels of Detail**

Imagina is probably one of the few systems that exhibits multiple levels of detail at all processing steps. A uniform design of the image processing algorithms integrates the level of detail seamlessly. Every algorithm can therefore be applied at multiple resolutions. This allows for noise tolerance when algorithms are applied at a coarse resolution. Multiple scales also allow for an early pruning of the search tree by first matching objects at a low resolution, and keeping only the best matches for applying finer-grain algorithms.

### **11.1.5 Explicitness of representation**

All processing stages and comparisons visually display why two regions were matched thus making the assumptions of the system explicit. Such an explicitness is crucial in understanding how the system works and thus being able to formulate more precise queries as well as understand how the system fails or why the system succeeds.

## **11.2 Shortcomings**

The main shortcoming of Imagina is that only color is used in segmenting images. Images which have little variety in the tones of color present are difficult to segment in a manner useful for the shape description process. One of the goals of Imagina is to be able to handle images of objects in uncontrolled situations. This requires that segmentation algorithms be very robust, and use as much information as possible from the image. Ideally, color, texture, shape, and shading could be used. Unfortunately, these features require progressively more effort and are less obvious to design.

Moreover, the shape matching relies uniquely on the color processing. The algorithms presented are robust, given a good color segmentation. Using other features of the image such as texture or edge detection can help make the segmentation more robust, and hence the entire system more reliable.

## **11.3 Conclusion**

As an image retrieval system, Imagina has been inspired heavily by the cognitive processes of visual recognition in biological systems. In doing so a uniform architecture has been designed where multiple methods are applied at multiple scales. This provides Imagina with a robustness against unexpected inputs, and a tolerance to noise.

Imagina, as a system is more geared towards light-duty tasks that can benefit from a better matching and interactivity at the cost of speed. As technology improves however, such systems will be more easily available for use in general tasks rather than fine-tuned recognition applications. This thesis has brought forward design principles to be used by image retrieval systems, and has summarized the elaborate literature in a few simple guidelines to follow in ulterior academic labor.

## Appendix A

# The Imagina System

The current implementation of the Imagina system is as a Java applet which can be run both through a browser and with an appletviewer. A client-server architecture using cgi scripts was never implemented. Therefore, the browser version is far more limited in its ability, as processing has to be done at the user's end, after a lengthy data transmission. Running the applet locally using an appletviewer, however, allows the use of the file system for rapid loading and saving of image objects.

The user front end of the Imagina system is the drawing interface. It allows for the creation of colored user sketches, providing both manual shape delineation and the use of primitives such as rectangles and ovals. The image which is drawn can either be saved for later review, or queried upon directly.

Segmenting the query is easy. The user can simply use as many colors as objects he is looking for. The segmentation is thus almost trivial on an image entirely constructed by the user.

However, the system also allows a user to bring in a query picture by simply specifying its URL. Hence any number of colors can be used in the query image, not all of them meaningful. However, as soon as we have to segment an imported image, we can just use the same algorithm as for database images.

### A.1 Implementation of Imagina as a Tool

The first part of this chapter introduced Imagina and the design principles we used to construct an image retrieval system that matches the human model of computation. Imagina is built primarily

as a proof of concept that image retrieval can be based upon a cognitive abstraction architecture.

However, from a more practical standpoint, Imagina was implemented to be a tool we will use to retrieve images from our own collections. A complementary set of implementation principles was used in building the system. These goals overlap with our theoretical design principles, and complement them in defining Imagina as a system.

- The query can be any image downloaded from the web
- No computational bottlenecks are introduced in matching
- An extensible architecture allows system evolution
- Algorithmic modules can be easily replaced and upgraded
- The inputs and outputs of each module can be displayed on screen
- Platform and Display Independence make the system portable

## A.2 Sketch and Image Upload

What sets Imagina apart from most current content-based image retrieval work is the sketch interface for formulating queries. As part of Imagina, a simple drawing program allows users to draw colored sketches describing regions.

Not only sketches but also images from any URL can be used in defining a query. The user can also import images by providing a URL, and later modify them to fit the needs of a particular query. This allows for a quick and easy query mechanism. A picture of the Eiffel Tower can be found on the web, edited within the sketch interface to erase the top, and then input to Imagina to find images of the construction of the Eiffel Tower.

What sets Imagina apart from most current content-based image retrieval work is the sketch interface for formulating queries. Computation on the query is cheap, and can be easily compared with precomputed representations of the database images. Other systems can only query in terms of precomputed images within the database.

### **A.3 No Computational Bottlenecks**

Imagina is a prototype for a sketch-based image retrieval system. It is not designed to handle large numbers of images, nor is it designed for efficiency and speed of retrieval. However, design decisions were made such that no such computational bottlenecks exist in the process of querying the system. Thus the system can be later extended to become a commercial scale system.

The system is meant to work with a large number of users, issuing queries and retrieving results immediately. Therefore, only a short time is allotted to the system for the retrieval computation. Thus every operation that can be done off-line should be executed in advance, and only a small critical path of operations must be executed once the user issues the query.

This has moreover been achieved by the different windows of processing that can be used for pruning of bad matches even at a coarse description level.

### **A.4 Extensible Architecture**

Object oriented architecture abstracts away from our assumptions. Different displays can be added. The search engine can be made more efficient. Monitoring user activity can give feedback to the system. New functionality can be added without changes to the existing system.

The design of Imagina should be such that new algorithms can be seamlessly included, and old algorithms easily replaced. Further functionality can be added such as learning from user interaction or fast database access.

### **A.5 Easily replaced algorithmic modules**

The algorithms implementing the functionality in Imagina are not hardwired together, but instead plugged together in an image-interpretation pipeline. New modules can be added as processing stages, and existing modules can be easily replaced.

### **A.6 Explicitness of representation**

All processing stages and comparisons visually display why two regions were matched thus making the assumptions of the system explicit. Such an explicitness is crucial in understanding how the system works and thus being able to formulate more precise queries as well as understand how the

system fails or why the system succeeds.

## **A.7 Platform Independence**

Java was chosen to allow for platform independence. This would allow for the easy dispersement of the program, if so desired. The choice of Java has also aided in both the implementation of user interfaces, as well as in the design of a reusable and modular system.

## Appendix B

# The Algorithm Toolbox

A central part of the Imagina development was the building of an interactive tool for algorithm development and testing. By having a development tool separate from the target system, several algorithms could be developed and compared in parallel. Then, depending on performance, and on the necessity of a particular approach, these algorithms could be introduced into the Imagina image search engine.

The testing program consists primarily of two images which are loaded, one on each side, and a set of buttons which allow the application of different algorithms lined down the middle of the interface. Because of its visual setup, this forced the development of visualization methods for almost every algorithm developed. Thus, the final database and query interface were easily implemented using these informational outputs for displaying match information.



## Appendix C

### Altavista's Image Finder

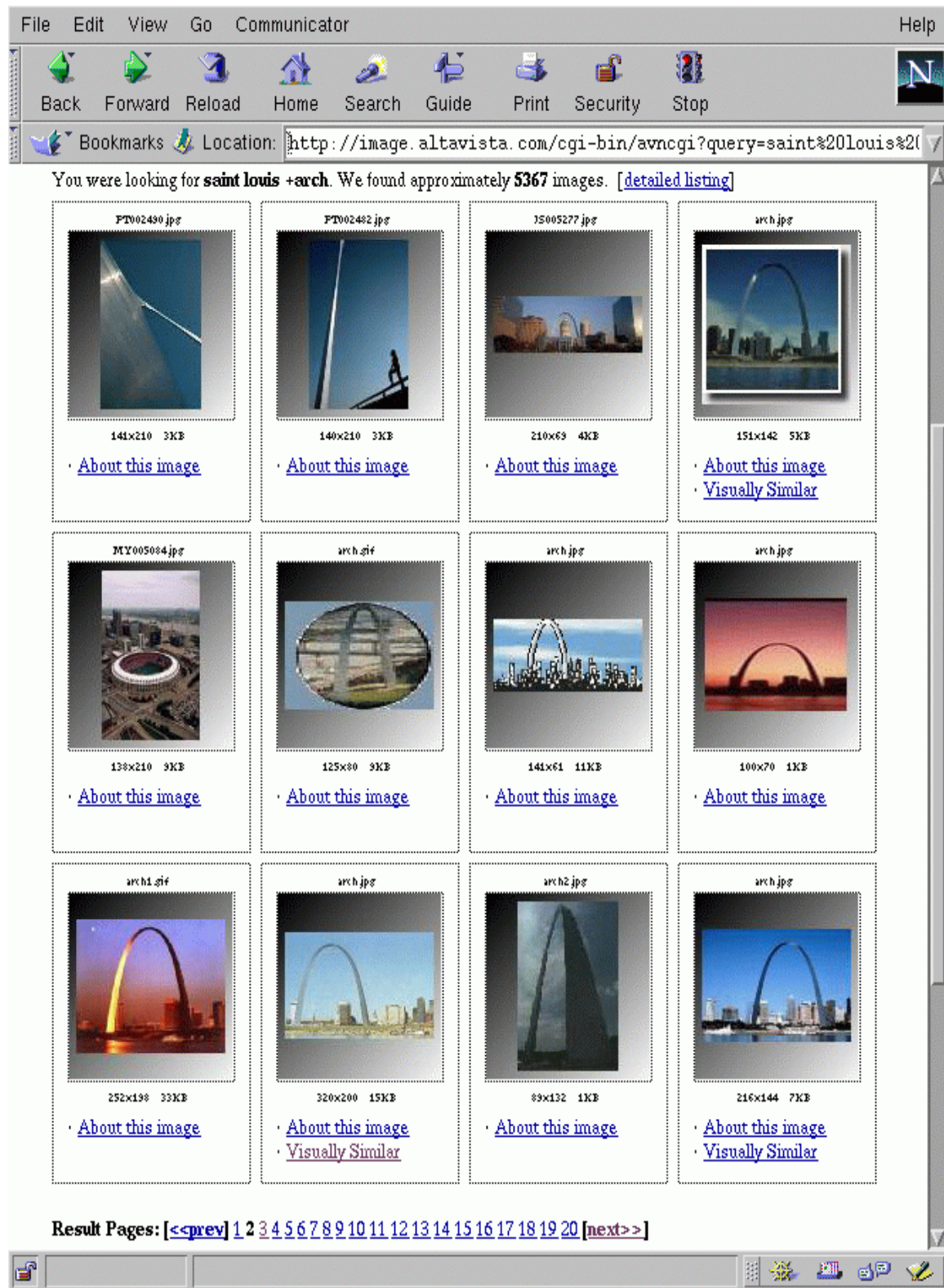


Figure C-1: A text-based search can be very effective when looking for a particular object

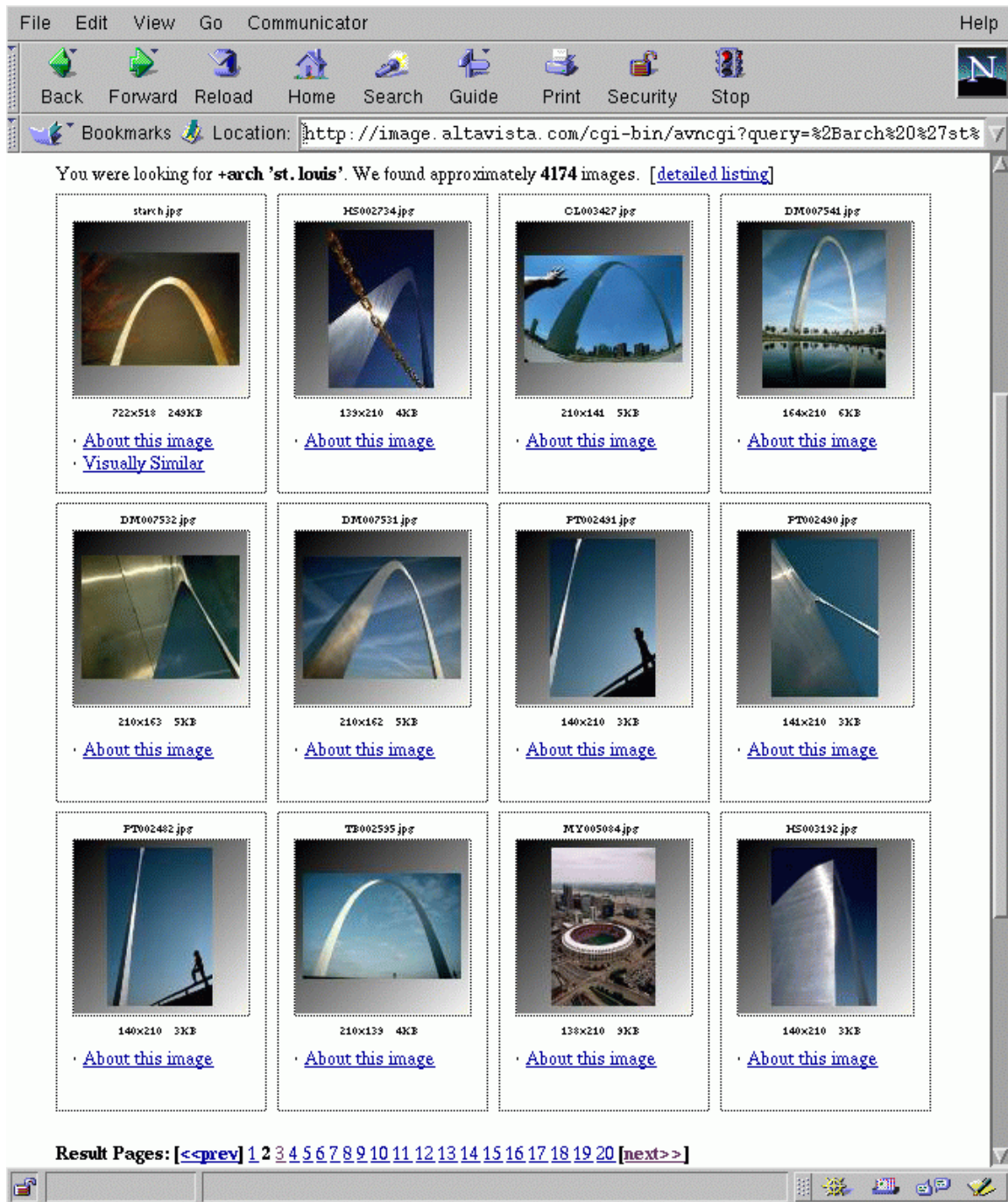


Figure C-2:



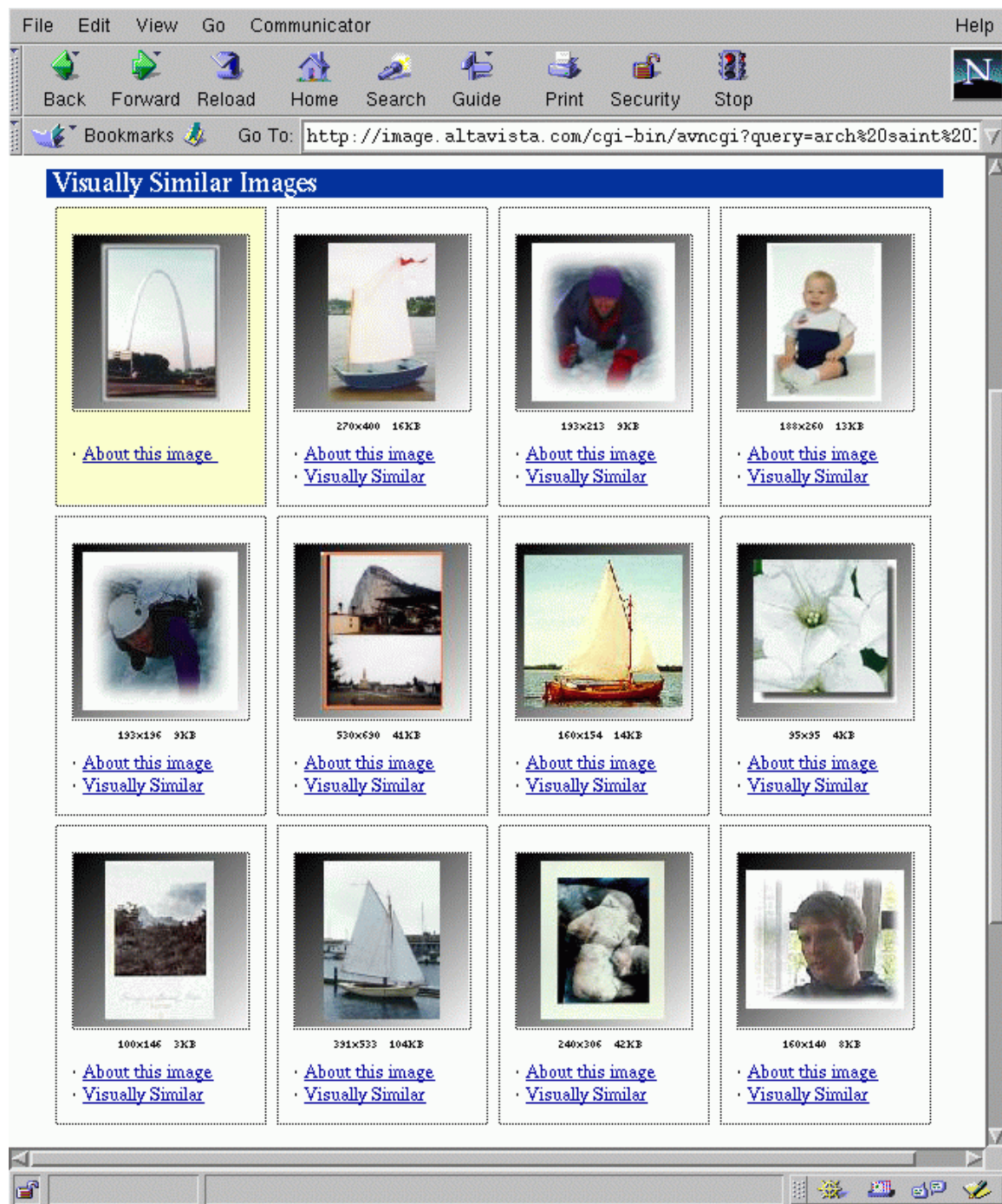


Figure C-3: Altavista Image Finder. The performance of an image search engine that relies mainly on color is very poor.

## Appendix D

# Image User Queries and Matches

The queries presented here were run on a single database of images. These samples were created by a single user, Patrycja Missiuro, after viewing a larger set of sample images which were used during the implementation of the Imagina system. The sample database that the queries were run on was then selected such that images which were similar to each of the queries were always present.

The size of the database of images unfortunately had to be limited because of main memory limitations. The main problem was the storage and allocation of displays which provided user information. Optimizations could have been implemented, for example by limiting the size of the internal display storage to the size of the screen display. However, due to time constraints this was not done.

The queries presented to the system are shown in Figures D-1 and D-2. The images were drawn using the Imagina query drawing tool, and were first saved as GIF images before being inserted into the database for indexing. The set of query matches presented here are the results of whole-image queries. Partial-image queries in the form of region queries are presented in a subsequent section.

The figures following first depict the user queries, then the database images which they were compared against, and lastly the matches determined by the system for each user query.



Figure D-1: The first three user queries.



Figure D-2: The second set of three user queries.



Figure D-3: The first set of database images.

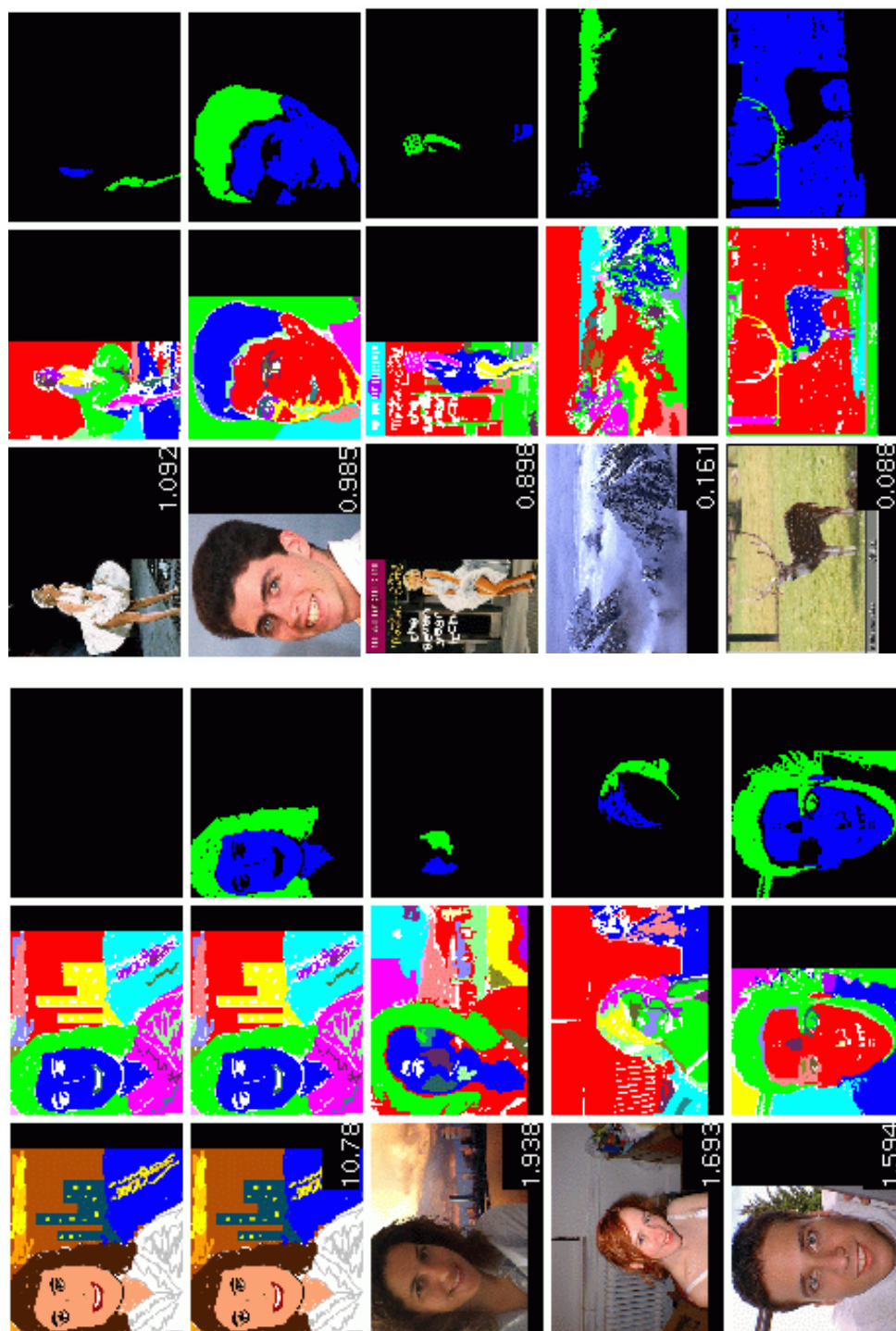


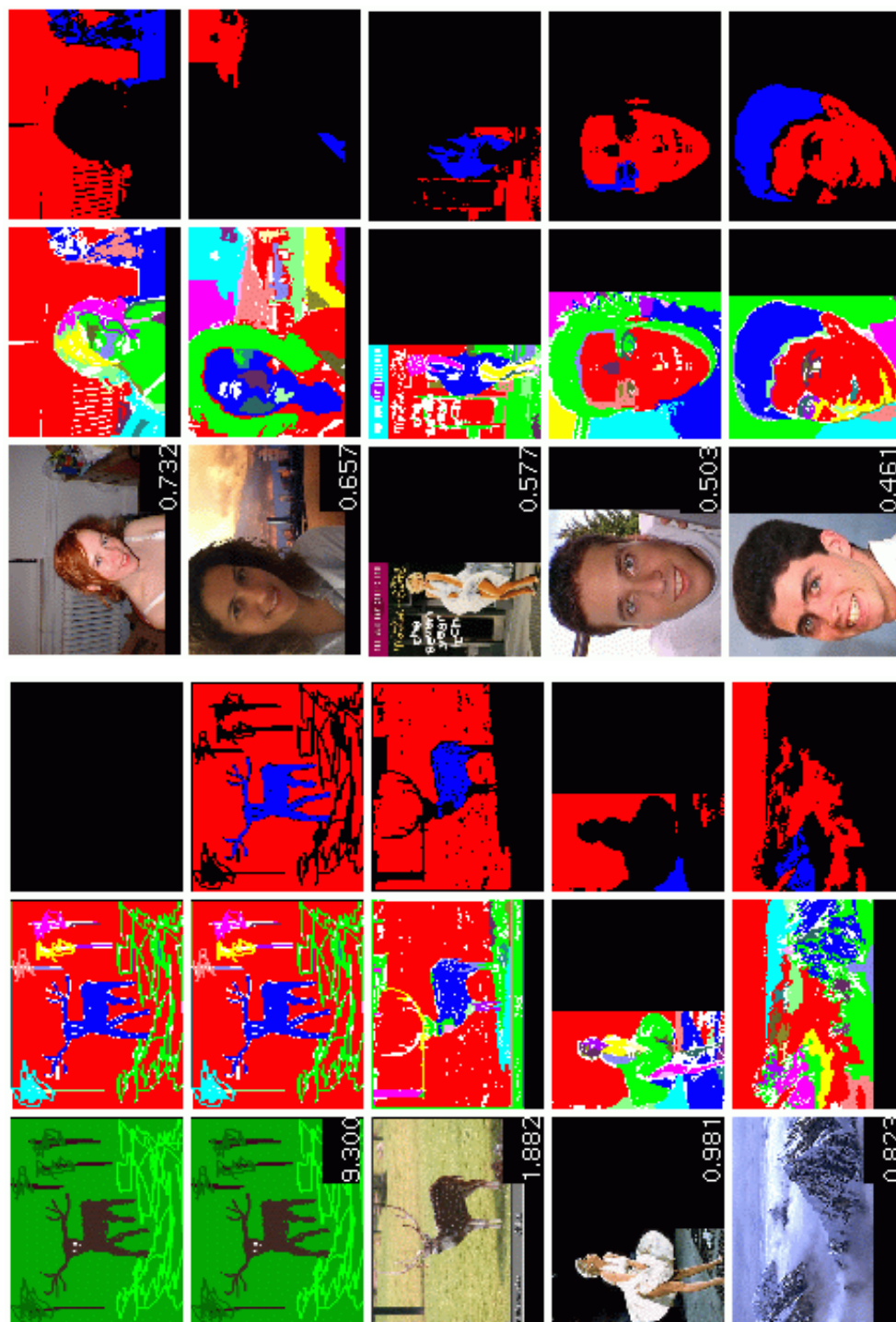


Figure D-4: The second set of database images.



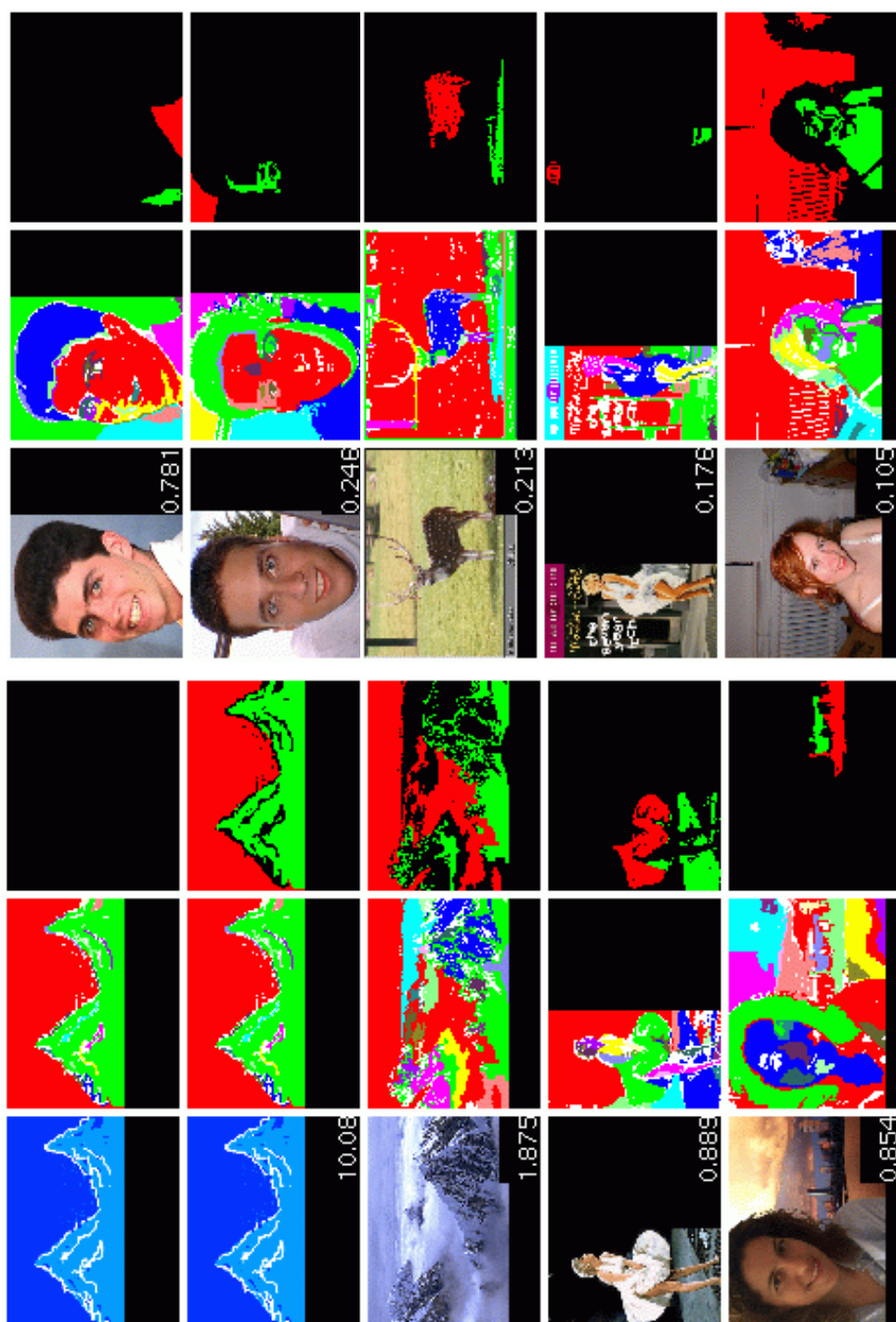
Figure D-5: The third set of database images.

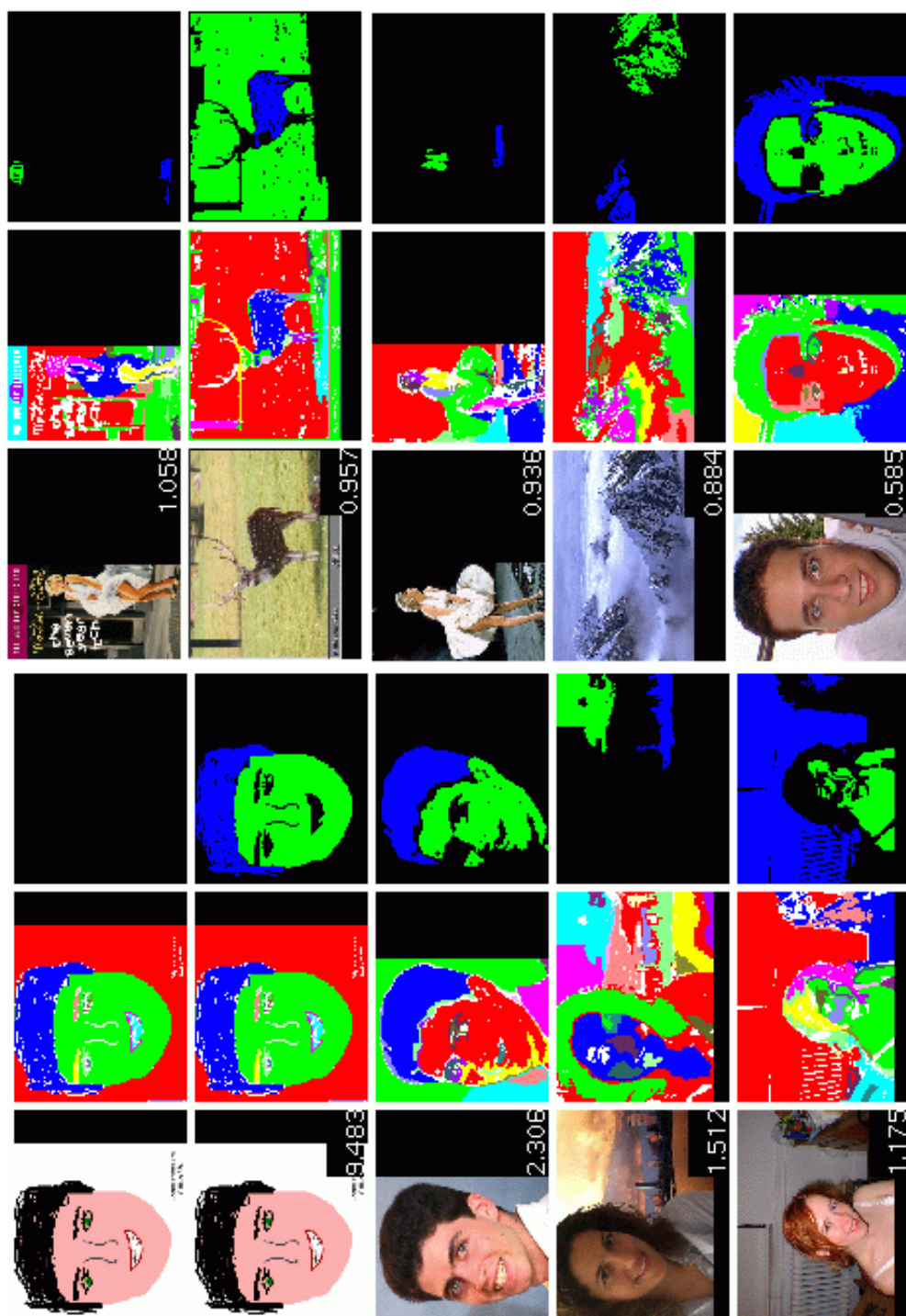


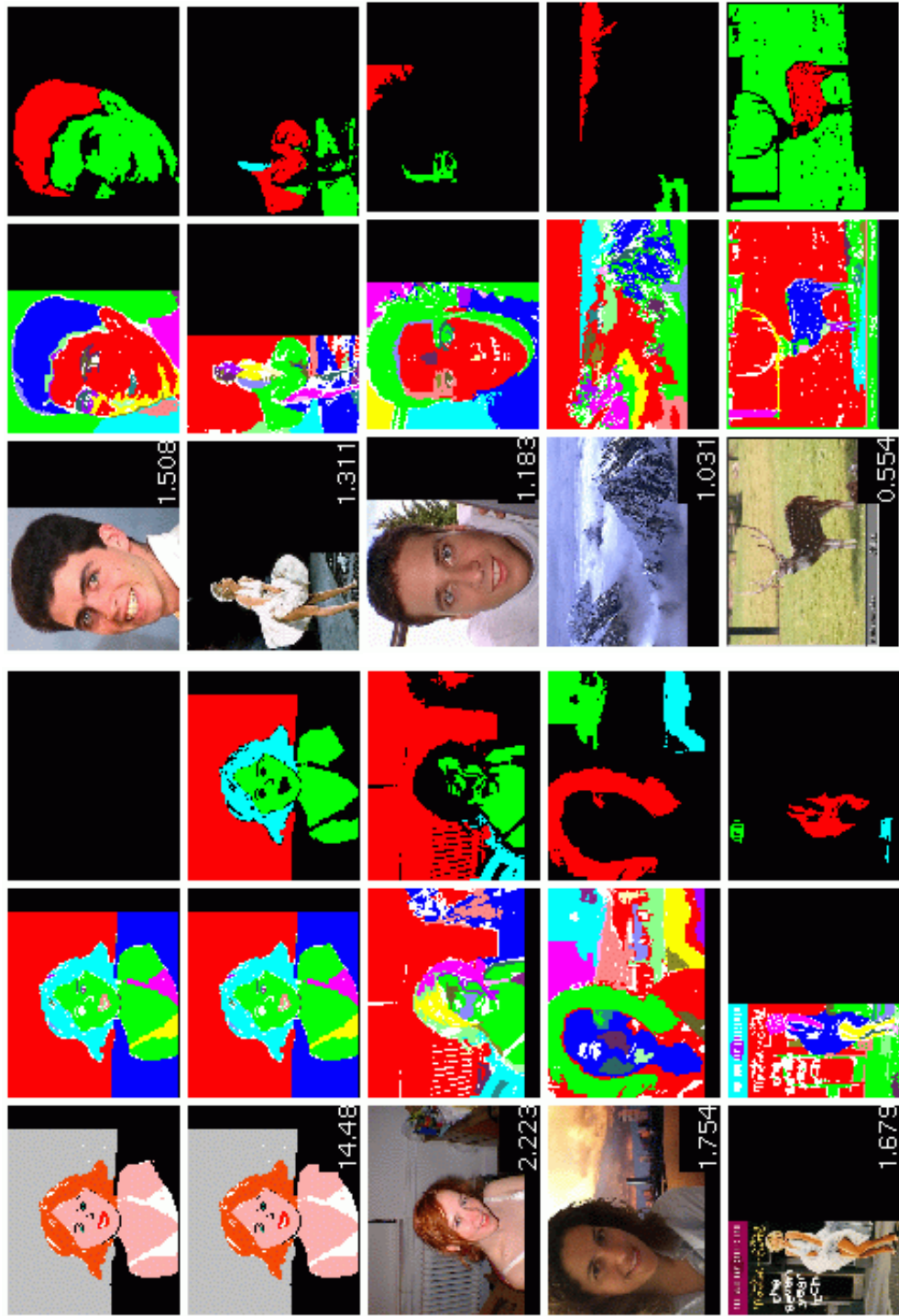














## Appendix E

# Image User Queries and Matches

The queries listed here are based on the same query set as the queries in appendix D. These queries highlight the methodology of region matching.

Two search screens are shown at nearly full size, and a third is shown shrunk. Each of these displays a single query result. Although the query results are multiple pages long, only the top matches are displayed in this appendix.

In the search screen, the query region is shown in the first row of the display. The regions which match the best are listed in sequential order, from best to worst, below this. Because the image which contains the query region was inserted into the database before searching in both cases, the query region also shows up as its own best match. This is a good sanity check of the functionality of the comparison mechanisms employed.

Two color and three shape matching metrics are used for region comparison. These are, in order from left to right: color histograms, color mappings, volume-based representation, segment-based representation, and angle-based representation.

The top value displayed in each screen field is the computed match of that region with the query region based on the given representation. The second value is the normalized value for the given representation space. The goodness value of each region is the weighted sum of these values. For the queries shown, the default weights of 1 for every representation space were used.















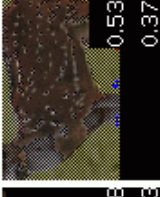


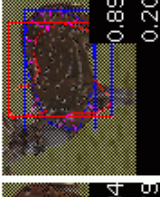
Applet

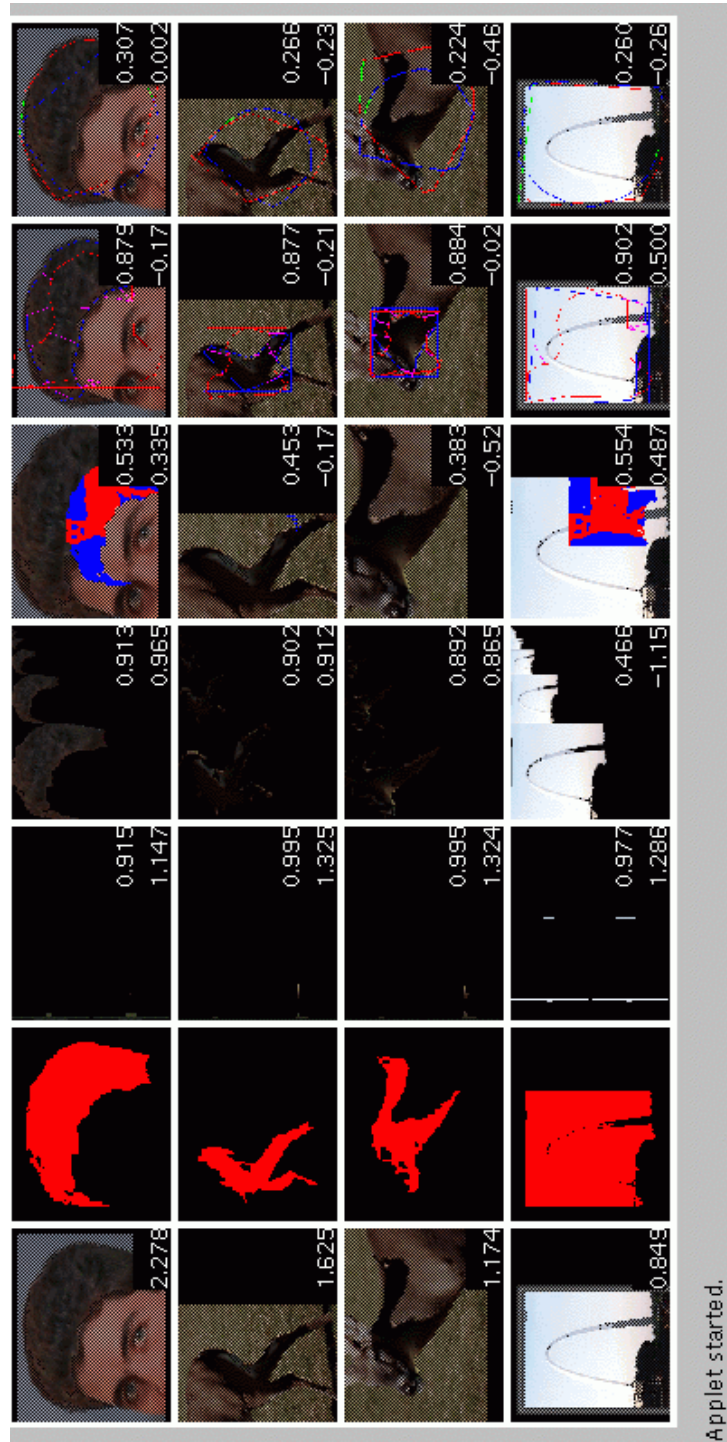
Mouse Click Mode:  Region Number Limit For Image Indexing:  Debug:

Load Image:

File Image:

Weightings for each Rep:

																	
13.20	2.701																




















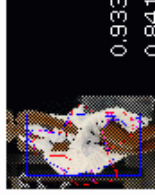
Applet

Mouse Click Mode:  Region Number Limit For Image Indexing:  Debug:

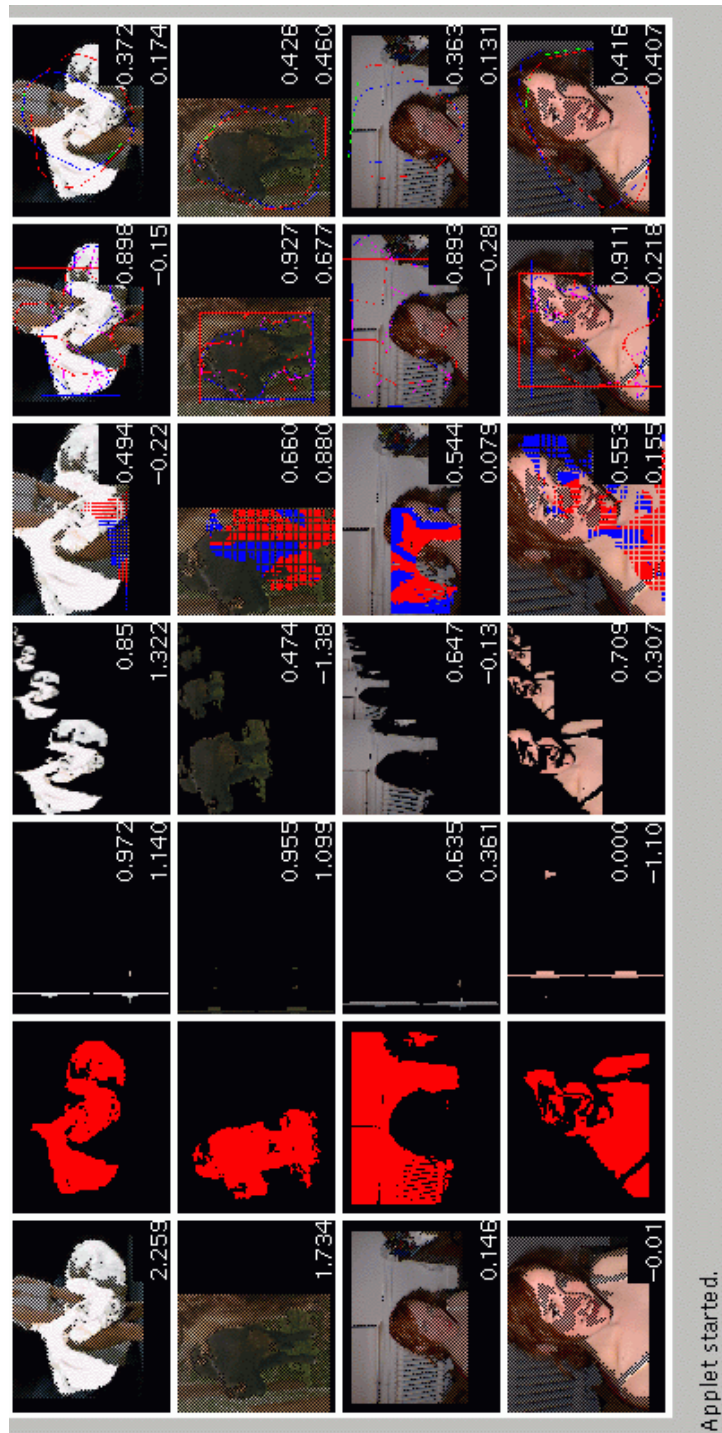
Load Image:

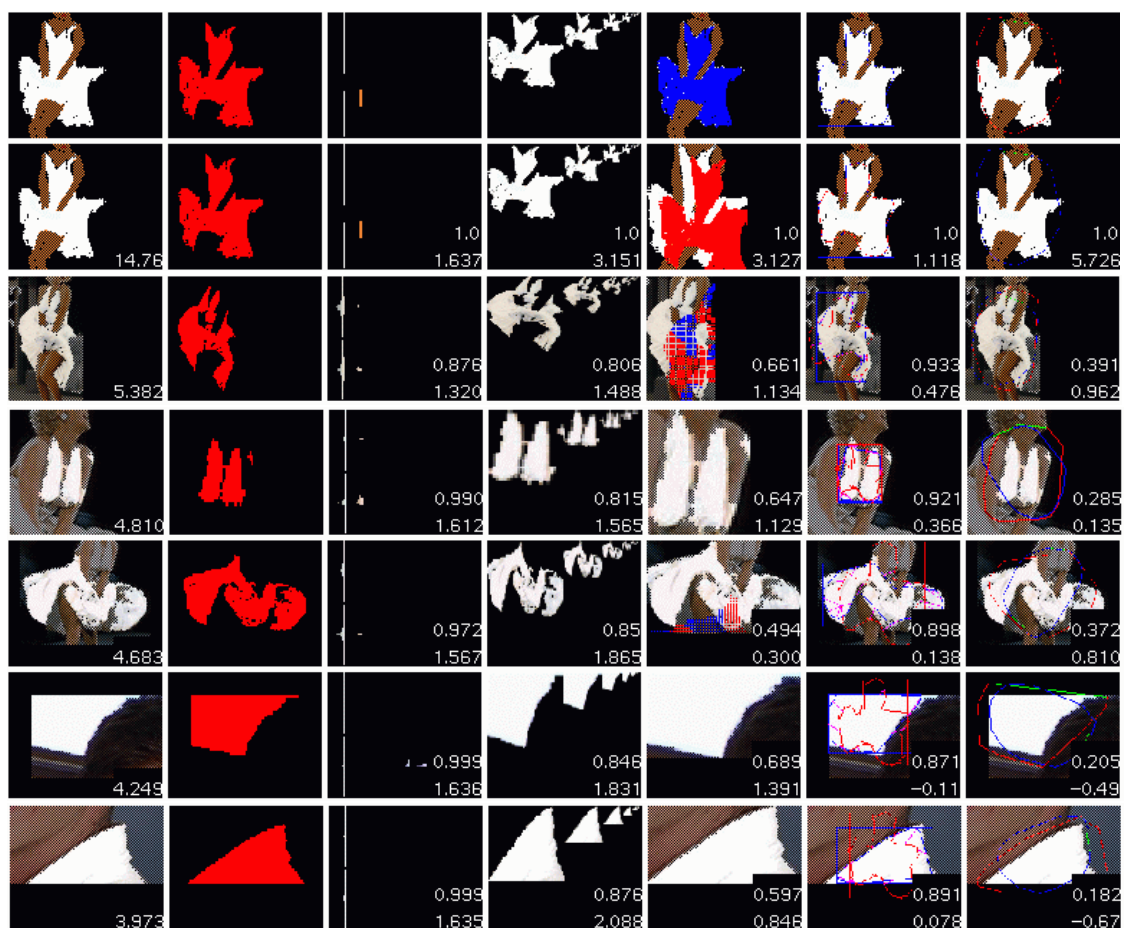
File Image:

Weights for each Rep:

																	
13.03																	
3.837																	







# Bibliography

- [1] Nina Amenta, Marshal Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. *ACM Siggraph*, August 1998. A provably correct algorithm for reconstructing the shape of an object from samples on its boundary.
- [2] Serge Belongie, Chad Carson, Hayit Greenspan, and Jitendra. Malik. Color- and texture-based image segmentation using em and its application to content-based image retrieval. *International Conference on Computer Vision (ICCV) '98*, 1998. This is the short version of [5], in which it was included almost verbatim.
- [3] Egon Brunswik and Joe Kamiya. Ecological cue-validity of “proximity” and of other gestalt factors. *American Journal of Psychology*, 66:20–32, 1953. Looks at whether Gestalt principles apply to real image contexts by looking at some film sections.
- [4] H. H. Bülthoff, S. Y. Edelman, and M. J. Tarr. How are three-dimensional objects represented in the brain? Technical Report 5, Max-Planck-Institut für biologische Kybernetik, Tübingen, Germany, 1994. Also published in 1995 in *Cerebral Cortex*, 5(3), pp. 247-260. This paper is among several published by these three authors, in combination or separately, in the early 90’s, on the topic of whether visual objects are stored with 2-D or 3-D representations in the human brain.
- [5] Chad Carson, Serge Belongie, Hayit Greenspan, and Jitendra. Malik. Color- and texture-based image segmentation using em and its application to image querying and classification. *Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1999. This paper is in review as of the time of the writing. It is an in-depth description of the Blobworld content-based image indexing tool.

- [6] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein, and Jitendra Malik. Blobworld: A system for region-based image indexing and retrieval. *Visual Information Systems '99*, 1999. One of several papers describing the Blobworld image indexing system.
- [7] P. S. Churchland and T. J. Sejnowski. *The Computational Brain*. MIT Press, Cambridge, Massachusetts, 1992. A computational look at neuroscience.
- [8] David A. Forsyth and Margaret M. Fleck. Body plans. *Computer Vision and Pattern Recognition (CVPR) '97*, Puerto Rico, June 1997. This paper is related to the Blobworld system. It describes the thinking which later went into building the system.
- [9] David A. Forsyth, Jitendra Malik, Margaret M. Fleck, Hayit Greenspan, Thomas Leung, Serge Belongie, Chad Carson, and Chris Blegler. Finding pictures of objects in large collections of images. *European Conference on Computer Vision (ECCV) '96 Workshop on Object Recognition for Computer Vision.*, 1996. This paper is related to the Blobworld system. It describes the thinking which later went into building the system.
- [10] James J. Gibson. Observations on active touch. *Psychological Review*, 69(6):477–491, 1962. A very thought-provoking paper on the difference between actively seeking out sensory input and just receiving sensory input. The argument is that the only way to really get a sense of the world is by the former, not the latter. A must-read.
- [11] L. Van Gool, J. Wagemans, J Vandeneede, and A. Oosterlinck. Similarity extraction and modeling. *IEEE International Conference on Computer Vision*, pages 530–534, 1990. A discussion of computing similarity between shapes based on their points of inflection and of zero curvature.
- [12] A. Gupta and R. Jain. Visual information retrieval. *Communications of the Association for Computing Machinery*, 40(5):70–79, 1997. This paper discusses image transformation techniques, limited applications and data query methods.
- [13] B. K. P. Horn. *Robot Vision*. MIT Press, Cambridge, Massachusetts, 1986. **The** classic book on computational approaches to vision, with a very strong mathematical handling of the subject.



- [14] B. K. P. Horn and B. G. Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981. This is one of the most well-known descriptions of the computations required by an optical flow algorithm. It is cited even in recent publications on the topic.
- [15] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(11):1254–1259, 1998. A model based on the early visual system in primates. The focus of interest is the issue of how to compute the salient locations in an input image. In primate vision, this is where attention is drawn and therefore potentially the section which should be segmented with the most care in an image indexing system.
- [16] D. J. Jobson, Z. Rahman, and G. A. Woodell. A multi-scale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing: Special Issue on Color Processing*, July 1997. This article discusses an application based on the retinex theory which allows a nearly illumination-invariant description of an image. This article glosses over most implementation details.
- [17] D. J. Jobson, Z. Rahman, and G. A. Woodell. Properties and performance of a center/surround retinex. *IEEE Transactions on Image Processing*, March 1997. The retinex theory as applied to image color enhancement is discussed. Although a potentially useful image pre-processing step, this was not applied in the Imagina system.
- [18] Manolis Kamvysselis. Two-dimensional polygon morphing using the extended gaussian image. <http://mit.edu/manoli/ecimorph/>, December 1997. Morphs two polygons by mapping edges to weights on the unit circle and transforming one set of weights to another.
- [19] Panayiotis Kamvysselis. Recursive expectation-maximization, an algorithm for segmentation. Masters thesis, Massachusetts Institute of Technology (MIT), Department of Electrical Engineering and Computer Science, June 1998. A model-based segmentation algorithm for recursive Expectation-Maximization (EM). Demonstrates superior results to prior EM algorithms. It is guaranteed to terminate, is deterministic, and requires no prior knowledge of the number of classification clusters.
- [20] P. J. Kellman and M. E. Arterbury. *The Cradle of Knowledge*. MIT Press, Cambridge, Massachusetts, 1998. An in-depth look at the experimental cognitive science literature on

infant cognition. This book is a good overview. Several chapters discuss object and scene recognition, directly and indirectly.

- [21] P. J. Kellman and E. S. Spelke. Perception of partly occluded objects in infancy. *Cognitive Psychology*, 15:483–524, 1983. Addresses the question of when infants consider a display to contain one object, and when infants consider a display to contain two objects, given that the connection between two visible parts of some object is hidden. The bottom line: motion, not color and form, are used in object segmentation early on.
- [22] Robert Laganière. Morphological corner detection. In *6th International Conference on Computer Vision*, pages 280–285, Bombay, India, January 1998. IEEE, Narosa Publishing House. A paper describing a corner detection algorithm based on a form of template matching called mathematical morphology.
- [23] Edwin Land. An alternative technique for the computation of the designator in the retinex theory of color vision. In *Proceedings of the National Academy of Science*, pages 3078–3080, 1986. This is one of Land’s later papers on retinex. This reference is included as a starting point for readers who may be more familiar with his earlier writings on the topic.
- [24] Alès Leonardis, Alok Gupta, and Ruzena Bajcsy. Segmentation as the search for the best description of the image in terms of primitives. *IEEE International Conference on Computer Vision*, pages 121–125, 1990. An older paper on implementations of image segmentation.
- [25] Pamela R. Lipson. Context and configuration based scene classification. Phd thesis, Massachusetts Institute of Technology (MIT), Department of Electrical Engineering and Computer Science, 1996. This thesis is related to the topic presented in Chapter 7 of Pawan Sinha’s PhD thesis[39]. It discusses qualitative image descriptions more in depth, culminating with the use of templates to do image classification.
- [26] Thomas Minka. An image database browser that learns from user interaction. Technical Report 365, MIT Media Laboratory, 1996. Describes the Photobook image indexing tool in great detail. This report is also listed as an MIT thesis for the degree of Master of Electrical Engineering and Computer Science.
- [27] W. Niblack, R. Barber, W. Equitz, M. Flickner, D. Glasman, D. Petkovic, and P. Yanker. The qbic project: Querying images by content using color, texture and shape. *SPIE Proc. Storage*

- and Retrieval for Image and Video Databases*, pages 173–187, 1993. One of the many papers describing IBM’s QBIC project.
- [28] R. Parasuraman, editor. *The Attentive Brain*. MIT Press, Cambridge, Massachusetts, 1998. Neuroscience with a focus on the universality of attention.
  - [29] Zoran Pečenović, Minh Do, Serge Ayer, and Martin Vetterli. New methods for image retrieval. In *Proceedings of the International Congress on Imaging Science*, September 1998. A summary of most recent approaches to content-based image indexing. A good starting point on current literature on the topic.
  - [30] Alex Pentland. Wearable intelligence. *Scientific American*, 276(1), 1998. A general overview of his work. This paper contains no useful technical specifics.
  - [31] Alex Pentland, Rosalind W. Picard, and Stan Sclaroff. Photobook: Tools for content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, 1996. An overview of Photobook, which is a system for image storage, indexing and retrieval, with some technical details.
  - [32] M. A. Peterson and B. S. Gibson. Object recognition contributions to figure-ground organization: Operations on outlines and subjective contours. *Perception and Psychophysics*, 56(5):551–564, 1994. This paper looks at whether outlines and subjective contours enable figure-ground perception. They argue that object recognition is based on edges determined during early visual processing.
  - [33] Photo and Media Finder of AltaVista. <http://image.altavista.com/>, 1998. An image searching and thumbnail preview front end to AltaVista.
  - [34] Aparna Lakshmi Ratan. The role of fixation and visual attention in object recognition. Technical Report 1529, MIT Artificial Intelligence Laboratory, January 1995. Although fixation and visual attention are interesting topics, the discussion of the use of color for image segmentation in chapter 3 of the report proved to be of most use to the Imagina project.
  - [35] M. D. Rugg, editor. *Cognitive Neuroscience*. MIT Press, Cambridge, Massachusetts, 1997. A neuroscience textbook: a must-have for a feel of the current understanding of brain functioning.

- [36] Erez Sali and Shimon Ullman. Recognizing novel 3-d objects under new illumination and viewing position using a small number of example views or even a single view. In *6th International Conference on Computer Vision*, pages 153–161, Bombay, India, January 1998. IEEE, Narosa Publishing House. This paper presents a method for class recognition based on a few example views under different conditions. Class-based generalization is presented for both viewing position and illumination changes.
- [37] Kaleem Siddiqi, Ali Shokoufandeh, Sven J. Dickinson, and Steven W. Zucker. Shock graphs and shape matching. In *6th International Conference on Computer Vision*, pages 222–229, Bombay, India, January 1998. IEEE, Narosa Publishing House. This paper is a follow-up to [38]. In it the authors propose the usage of the shock graph description for 2-D shape matching.
- [38] Kaleem Siddiqi, Allen Tannenbaum, and Steven W. Zucker. Hyperbolic “Smoothing” of shapes. In *6th International Conference on Computer Vision*, pages 215–221, Bombay, India, January 1998. IEEE, Narosa Publishing House. This paper describes a method of modeling 2-D shapes which results in a series of descriptions which are similar to the different views of an object from several distances. The authors call this description a “shock graph”.
- [39] Pawan Sinha. Perceiving and recognizing three-dimensional forms. Phd thesis, Massachusetts Institute of Technology (MIT), Department of Electrical Engineering and Computer Science, 1995. Although very interesting, most of this thesis is not directly relevant to image indexing. Chapter 7, however, discusses a method of storing image descriptions in terms of the lightness ratios of neighboring patches in an image. The effectiveness of face detection using such a description, scaled to match, is presented.
- [40] John R. Smith. Integrated spatial and feature image systems: Retrieval, analysis and compression. Phd thesis, Columbia University, Graduate School of Arts and Sciences, 1997. This is an in-depth description of a VisualSeek, a content-based image retrieval system. Provides very good treatments of almost every related topic; for example, five different methods of representing color are described! Watch out for small errors in equations, however. Related works were published co-authored with his thesis supervisor, professor Shih-Fu Chang.
- [41] M. J. Swain and D. H. Ballard. Indexing via color histograms. *International Conference on Computer Vision (ICCV)*, 1990. Discusses how a properly described color space can be used for orientation- and illumination-invariant color-based image indexing.

- [42] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991. A more mature version of [41].
- [43] Georges Thines, Alan Costall, and George Butterworth, editors. *Michotte’s Experimental Phenomenology of Perception*. L. Erlbaum Associates, Hillsdale, N.J., 1991. This is a re-edition of Albert Michotte’s classic work. It can be found listed under either the editors’ names or under Michotte. Only the section on the amodal completion of perceptual structures was looked at.
- [44] A. Treisman and G. Gelade. A feature integration theory of attention. *Cognitive Psychology*, 12:97–136, 1980. Part of a long line of work by A. Treisman dealing with the pre-attentive selection of features.
- [45] Shimon Ullman. *High-Level Vision*. MIT Press, Cambridge, Massachusetts, 1996. Although it concentrates on Ullman’s work, it is a very good summary of the current state of understanding of human visual processes.
- [46] Francisco J. Varela, Evan Thompson, and Eleanor Rosch. *The Embodied Mind: Cognitive Science and Human Experience*. MIT Press, Cambridge, Massachusetts, 1991. The sections of interest are the ones on Rosch’s work in particular.
- [47] Virage. <http://www.virage.com/>, 1998. Virage Inc partnered with Altavista in providing the visual similarity search of the AltaVista Photo and Media Finder.
- [48] James Ze Wang, Gio Wiederhold, Oscar Firschein, and Sha Xin Wei. Wavelet-based image indexing techniques with partial sketch retrieval capability. In *Proceedings of the Fourth Forum on Research and Technology Advances in Digital Libraries (ADL’97)*, Washington, D.C., May 1997. A slightly different approach to content-based image indexing. As a bonus, wavelets allow compression of the images.
- [49] Max Wertheimer. *Readings in Perception, selected and edited by David C. Beardslee and Michael Wertheimer*, selection 8, pages 115–135. Van Nostrand, Princeton, N.J., 1958. This short paper, “Principles of Perceptual Organization”, provides a quick coverage of gestalt principles with respect to visual input.

COMPUTATIONAL COMPARATIVE GENOMICS:  
GENES, REGULATION, EVOLUTION

by

Manolis (Kellis) Kamvysselis

B.S. Electrical Engineering and Computer Science;  
M. Eng. Computer Science and Engineering  
Massachusetts Institute of Technology, 1999

Submitted to the Department of Electrical Engineering and Computer Science  
In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Computer Science

at the  
Massachusetts Institute of Technology  
June 2003

© 2003 Massachusetts Institute of Technology  
All rights reserved

Signature of Author .....  
Department of Electrical Engineering and Computer Science  
May 23, 2003

Certified by .....  
Eric S. Lander  
Professor of Biology  
Thesis Co-Supervisor

Certified by .....  
Bonnie A. Berger  
Professor of Applied Mathematics  
Thesis Co-Supervisor

Accepted by .....  
Arthur C. Smith  
Chairman, Committee on Graduate Students  
Department of Electrical Engineering and Computer Science



**Computational Comparative Genomics:  
Genes, Regulation, Evolution**

by

Manolis (Kellis) Kamvysselis

Submitted to the Department of Electrical Engineering and Computer Science  
on May 23, 2003 in Partial Fulfillment of the Requirements for the Degree of  
Doctor of Philosophy in Computer Science

**ABSTRACT**

Understanding the biological signals encoded in a genome is a key challenge of computational biology. These signals are encoded in the four-nucleotide alphabet of DNA and are responsible for all molecular processes in the cell. In particular, the genome contains the blueprint of all protein-coding genes and the regulatory motifs used to coordinate the expression of these genes. Comparative genome analysis of related species provides a general approach for identifying these functional elements, by virtue of their stronger conservation across evolutionary time.

In this thesis we address key issues in the comparative analysis of multiple species. We present novel computational methods in four areas (1) the automatic comparative annotation of multiple species and the determination of orthologous genes and intergenic regions (2) the validation of computationally predicted protein-coding genes (3) the systematic de-novo identification of regulatory motifs (4) the determination of combinatorial interactions between regulatory motifs.

We applied these methods to the comparative analysis of four yeast genomes, including the best-studied eukaryote, *Saccharomyces cerevisiae* or baker's yeast. Our results show that nearly a tenth of currently annotated yeast genes are not real, and have refined the structure of hundreds of genes. Additionally, we have automatically discovered a dictionary of regulatory motifs without any previous biological knowledge. These include most previously known regulatory motifs, and a number of novel motifs. We have automatically assigned candidate functions to the majority of motifs discovered, and defined biologically meaningful combinatorial interactions between them. Finally, we defined the regions and mechanisms of rapid evolution, with important biological implications.

Our results demonstrate the central role of computational tools in modern biology. The analyses presented in this thesis have revealed biological findings that could not have been discovered by traditional genetic methods, regardless of the time or effort spent. The methods presented are general and may present a new paradigm for understanding the genome of any single species. They are currently being applied to a kingdom-wide exploration of fungal genomes, and the comparative analysis of the human genome with that of the mouse and other mammals.

Thesis Co-Supervisor: Eric Lander, professor of Biology

Thesis Co-Supervisor: Bonnie Berger, professor of Applied Mathematics



## TABLE OF CONTENTS

OVERVIEW .....	7
Biological Signals.....	7
Contributions of this thesis.....	9
BACKGROUND .....	13
0.1. Molecular biology and the study of life. ....	13
0.2. Gene regulation and the dynamic cell .....	15
0.3. Evolutionary change and comparative genomics .....	17
0.4. Sequence alignment and phylogenetic trees.....	19
0.5. Model organisms and yeast genetics. ....	20
0.6. Genome sequencing and assembly.....	22
CHAPTER 1: GENOME CORRESPONDENCE .....	25
1.1. Introduction .....	25
1.2. Establishing gene correspondence.....	26
1.3. Overview of the algorithm .....	27
1.4. Automatic annotation and graph construction.....	28
1.5. Initial pruning of sub-optimal matches .....	30
1.6. Blocks of conserved synteny .....	30
1.7. Best Unambiguous Subsets .....	32
1.8. Performance of the algorithm.....	34
1.9. Conclusion.....	36
CHAPTER 2: GENE IDENTIFICATION.....	37
2.1. Introduction .....	37
2.2. Different conservation of genes and intergenic regions.....	38
2.3. Reading Frame Conservation Test .....	40
2.4. Results: Hundreds of previously annotated genes are not real.....	42
2.5. Refining Gene Structure.....	44
2.6. Analysis of small ORFs.....	48
2.7. Conclusion: Revised yeast gene catalog.....	50
CHAPTER 3: REGULATORY MOTIF DISCOVERY .....	51
3.1. Introduction .....	51
3.2. Regulatory motifs .....	52
3.3. Extracting signal from noise.....	54

3.4. Conservation properties of known regulatory motifs.....	55
3.5. Genome-wide motif discovery .....	58
3.7. Results and comparison to known motifs.....	63
3.8. Conclusion.....	64
CHAPTER 4: REGULATORY MOTIF FUNCTION .....	65
4.1. Introduction .....	65
4.2. Constructing functionally-related gene sets. ....	66
4.3. Assigning a function to the genome-wide motifs.....	67
4.4. Discovering additional motifs based on gene sets.....	71
4.7. Conclusion.....	74
CHAPTER 5: COMBINATORIAL REGULATION.....	75
5.1. Introduction .....	75
5.2. Motifs are shared, reused across functional categories .....	75
5.3. Changing specificity of motif combinations. ....	77
5.4. Genome-wide motif co-occurrence map. ....	78
5.5. Results. ....	79
5.6. Conclusion.....	80
CHAPTER 6: EVOLUTIONARY CHANGE.....	81
6.1. Introduction .....	81
6.2. Protein family expansions localize at the telomeres. ....	82
6.3. Chromosomal rearrangements mediated by specific sequences. ....	84
6.4. Small number of novel genes separate the species.....	85
6.5. Slow evolution suggests novel gene function. ....	86
6.6. Evidence and mechanisms of rapid protein change. ....	87
6.7. Conclusion.....	89
CONCLUSION.....	91
C.1. Summary.....	91
C.2. Extracting signal from noise. ....	92
C.4. The road ahead.....	94
REFERENCES .....	95
APPENDIX.....	100

## **ACKNOWLEDGEMENTS**

I am indebted to Eric Lander, Bonnie Berger and Bruce Birren for their constant help, advice, support, and mentorship in all aspects of my thesis and graduate career. Many thanks to my colleague Nick Patterson whose help and advice contributed to chapters 3 and 4, to David Gifford and Gerry Sussman for their advice, and to my friends Serafim Batzoglou, Sarah Calvo, James Galagan, Julia Zeitlinger for invaluable advice and support.

I would like to acknowledge the contribution of Matt Endrizzi and the staff of the Whitehead/MIT Center for Genome Research Sequencing Center, who generated the shotgun sequence from the three yeast species; David Botstein, Michael Cherry, Kara Dolinski, Diana Fisk, Shuai Weng and other members of the *Saccharomyces* Genome Database staff for assistance and discussions, and for making the data available to the community through SGD; Ed Louis and Ian Roberts who provided the yeast strains; Tony Lee, Nicola Rinaldi, Rick Young and the Young Lab for sharing data about chromatin immunoprecipitation experiments and for discussions; Michael Eisen and Audrey Gasch for sharing information about gene expression clusters and for discussions.

Many thanks to Gerry Fink, Martin Kupiec, Sue Lindquist, Andrew Murray, Heather True-Krobb for discussions and understanding of yeast biology. Many thanks to Jon Butler, Gus Cervini, Ken Dewar, Leslie Gaffney, David Jaffe, Joseph Lehar, Li Jun Ma, Abigail Melia, Chad Nusbaum and members of the WICGR for help and discussions.

I owe my gratitude to my parents John and Anna Kamvysselis, to my siblings Peter and Maria, and to Alexandra Mazalek for their love and constant support.

## OVERVIEW

### Biological Signals

Understanding the biological signals encoded in a genome is a key challenge of modern biology. These signals are encoded in the four-nucleotide alphabet of DNA and are responsible for all molecular processes in the cell. In particular, the genome contains the blueprint of all protein-coding genes and the control signals used to coordinate the expression of these genes. The well-being of any cell relies on the successful recognition of these signals, and a large number of biological mechanisms have evolved towards this goal. Specific protein complexes are responsible for the copying of a gene segment from DNA to messenger RNA (transcription) and for its eventual translation into protein following the genetic code to assign an amino acid to every tri-nucleotide codon. A specific class of proteins called transcription factors help recruit the transcription machinery to a target gene by binding their specific DNA signals (regulatory motifs) in response to environmental conditions. An abundance of information within the cell guides these processes, involving protein-protein and protein-DNA interactions between a multitude of players, the state of DNA coiling, and other mechanisms that are still not well-understood.

The computational identification of genes however, can only rely on the primary DNA sequence of the organism. Current programs use properties about the protein-coding potential of DNA segments that are unseen by the transcription machinery. In particular, since genes always start with an ATG (start codon) and end in with TAG, TGA, or TAA (one of three stop codons), programs exist that specifically look for these stretches between a start and a stop codon called ORFs (Open Reading Frames). The basic approach is to identify ORFs that are too long to have likely occurred by chance. Since stop codons occur at a frequency of 3 in 64 in random sequence, ORFs of 60 or even 150 amino acids will occur frequently by chance, but longer ORFs of 300 or thousands of amino acids are virtually always the result of biological selective pressure. Hence, simple computational programs can easily recognize long genes, but many small genes will be indistinguishable from spurious ORFs arising by chance. This is evidenced by the considerable debate over the number of genes in yeast<sup>1-5</sup> with proposed counts ranging from 4800 to 6400 genes. The situation is worse for organisms with large,

complex genomes, such as mammals where estimated gene counts have ranged from 30 to 120 thousand genes.

The direct identification of the repertoire of regulatory motifs in a genome is even more challenging. Regulatory motifs are short (typically 6-8 nucleotides), and do not obey the simple rules of protein-coding genes. In any single locus, nothing distinguishes these signals from random nucleotides. Traditionally, their discovery relied on deletion studies of consecutive DNA segments until regulation was disrupted and the control region was identified<sup>6</sup>. With the sequence of multiple genes in the same pathway at hand, it became possible to search for the repetition of these signals in genes controlled by the same transcription factor. Computational methods have been developed to search for enriched sequence motifs in predefined sets of genes (for example, using expectation-maximization<sup>7</sup> or gibbs-sampling<sup>8</sup>, reviewed in <sup>9</sup>). As microarray analysis provided genome-wide levels of gene expression under a various experimental conditions, computational methods of gene clustering have resulted in hundreds of such sets of genes. Various computational methods have been used to mine these sets for regulatory motifs, and dozens of candidate motifs have resulted from each search. The vast majority of these candidate motifs are due to noise however, and only a total of about 50 real motifs have currently been discovered.

The current methods of motif identification suffer from a number of limitations. (a) First and foremost is that the weak signal of small motifs is hidden in the noise of relatively large intergenic regions. This inherent signal to noise ratio limits even the best programs from recognizing true motifs in the input data. (b) Additionally, the sets of genes searched, and hence the motifs discovered, are limited by our current biological knowledge of co-regulated sets of genes. The current knowledge is based on the experimental conditions reproduced in the lab, which is likely to be a small fraction of the vast array of environmental responses yeast uses to survive in its natural habitat. (c) Finally, an emerging view of gene regulation has put in question the approaches that search for a single motif responsible for a pathway or environmental response. Pathways are not regulated as isolated components in the cell. Genes and transcription factors have multiple functions and are used in multiple pathways and environmental responses. More importantly, transcription factors do not act in isolation, and protein-protein interactions

between factors are as important as protein-DNA interactions between each individual factor and its target genes. Hence, individual gene sets will be enriched in multiple motifs, and individual motifs will be enriched in multiple gene sets. A comprehensive understanding of regulatory motifs requires a novel, more powerful approach.

Comparative genome analysis of related species should provide such a general approach for identifying functional elements without prior knowledge of function. Evolution relentlessly tinkers with genome sequence and tests the results by natural selection. Mutations in non-functional nucleotides are tolerated and accumulate over evolutionary time. However, mutations in functional nucleotides are deleterious to the organism that carries them, and become sparse or extinct. Hence, functional elements should stand out by virtue of having a greater degree of conservation across the genomes of related species. Recent studies have demonstrated the potential power of comparative genomic comparison. Cross-species conservation has previously been used to identify putative genes or regulatory elements in small genomic regions<sup>10-13</sup>. Light sampling of whole-genome sequence has been used as a way to improve genome annotation<sup>4,14</sup>. Complete bacterial genomes have been compared to identify pathogenic and other genes<sup>15-18</sup>. Genome-wide comparison has been used to estimate the proportion of the mammalian genome under selection<sup>19</sup>.

### **Contributions of this thesis**

The goal of this thesis is to develop computational comparative methods to understand genomes. We develop and apply general approaches for the systematic analysis of protein-coding and regulatory elements by means of whole-genome comparisons with multiple related species. We apply these methods to *Saccharomyces cerevisiae*, commonly known as baker's yeast. *S. cerevisiae* is a model organism for which many genetic tools and techniques have been developed, leading to a wealth of experimental information. This knowledge has allowed us to validate our biological predictions and assess the power of the methods developed. We generated high-quality draft genome sequences from three *Saccharomyces* species of yeast related to *S. cerevisiae*. These data provide us with invaluable comparative information currently unmatched by previous sequencing efforts. Starting with the raw nucleotide sequence assemblies of the three newly sequenced species and the current sequence and annotation

of *S. cerevisiae*, we set out to discover functional elements in the yeast genome based on the comparison of the four species.

We first present methods for the automatic comparative annotation of the four species and the determination of orthologous genes and intergenic regions (Chapter 1). The algorithms enabled the automatic identification of orthologs for more than 90% of genes despite the large number of duplicated genes in the yeast genome.

Given the gene correspondence, we construct multiple alignments and present comparative methods for gene identification (Chapter 2). These rely on the different patterns of nucleotide change observed in the alignments of protein coding regions as compared to non-coding regions, specifically the pressure to conserve the reading frame of proteins. The method has high specificity and sensitivity, and enabled us to revisit the current gene catalogue of *S. cerevisiae* with important biological implications.

We then turn to the identification of regulatory motifs (Chapter 3). We present statistical methods for their systematic de-novo identification without use of prior biological information. We automatically identified 72 genome-wide sequence elements, with strongly non-random conservation properties. To validate our findings, we compared the discovered motifs against a list of known motifs, and found that we discovered virtually all previously known regulatory motifs, and an additional 41 motifs. We assign function to these motifs using sets of functionally related genes (Chapter 4), and we discover additional motifs enriched in these sets.

We further present methods for revealing the combinatorial control of gene expression (Chapter 5). We study the genome-wide co-occurrence of regulatory motifs, and discover significant correlations between pairs of motifs that were not apparent in a single genome. We show that these correspond to biologically meaningful relationships between the corresponding factors and that motif combinations can change the specific functional enrichment of target genes, thus increasing the versatility of gene regulation using only a limited number of regulatory motifs.

We finally focus on the differences between the species compared and discover the regions and mechanisms of evolutionary change (Chapter 6). We study rapid gene family expansions and discover that they localize in the telomeres. We show that chromosomal rearrangements and inversions are mediated by specific sequence elements.

We find specific mechanisms of rapid protein change in environment adaptation genes, as well as stretches of unchanged nucleotides suggesting novel functions for uncharacterized genes.

Our results demonstrate the central role of computational tools in modern biology. Our methods are general and applicable to the study of any organism. They are currently being applied to a kingdom-wide exploration of fungal genomes and the comparative analysis of the human genome with that of the mouse and other mammals. Comparison of multiple related species may present a new paradigm for understanding the genome of any single species.





## BACKGROUND

### 0.1. Molecular biology and the study of life.

It is both humbling and bewildering that what separates humans from bacteria is merely the organization and assembly of the same basic bio-molecules. It is the study of these shared foundations of life that gave rise to the discipline of *molecular biology*. In the microscopic level, complex and simple organisms alike are made up of the same unit of life, the *cell*. A cell contains all the information and machinery necessary for its growth, maintenance and replication. It is delimited from its surrounding by a water-impermeable membrane and all communication and transport across the membrane is tightly controlled. Two major types of cells exist, *prokaryotic* cells with simple internal organization, and *eukaryotic* cells, with extensive compartmentalization of functions such as information storage in the nucleus, energy production in mitochondria, metabolism in the cytoplasm, etc. In unicellular organisms, the cell constitutes the complete organism, whereas multi-cellular organisms (typically eukaryotes) can contain up to trillions of cells, and hundreds of specialized cell types. In either case though, a cell can rarely be thought of in isolation, but is constantly interacting with its surrounding, sensing the presence of environmental changes, and exchanging stimuli with other cells that may be part of the same colony or organism.

Within a cell, virtually all functional roles are fulfilled by *proteins*, the most versatile type of macromolecule. Various types of proteins fulfill an immense array of tasks. For example, enzymes catalyze countless chemical reactions; transcription factors control the timing of gene usage; transporters carry molecules inside or outside the cell; trans-membrane channels regulate the concentrations of molecules in the cell; structural proteins provide support and shape to the cell; actins can cause motion; receptors recognize intra- or extra-cellular signals. This incredible versatility of proteins comes from the innumerable combinations of an alphabet of only 20 *amino acid* building blocks, juxtaposed in a single unbranched chain of hundreds or thousands of such amino acids. All amino acids share an identical portion of their structure that forms the protein backbone, to which is attached one of 20 possible side chains of variable size, shape, charge, polarity, hydrophobicity. The precise sequence of amino acids dictates a unique

three-dimensional fold that optimizes electrostatic and other interactions between the side-chains and with the solvent.

*DNA* in turn carries the genetic information that encodes the precise sequence of all proteins, the signals that control their production, and all other inheritable traits. DNA is also a macromolecule, consisting of the linear juxtaposition of millions of *nucleotides*. It encodes the genetic information digitally, like the bits of a digital computer, in the precise ordering of four types of nucleotides. Like amino-acids, these nucleotides share a fixed portion that forms a (phosphate) backbone to which is connected (via a deoxyribose sugar) a variable portion that is one of four *bases*, abbreviated A, C, G, T. Unlike proteins however, the structure of DNA is fixed. It consists of two strands, like the sidepieces of a ladder, connected by pairs of bases, like the steps of ladder. The two strands are wrapped around each other and form a double-helix. The two phosphate backbones form the outside of the helix, and the base pairs, connected by weak hydrogen bonds, form the interior of the helix. Only two pairings of bases are possible, based on shape and charge complementarity: A always pairs with T and C always pairs with G. This self-complementarity of the DNA structure forms the very basis of heredity: during *DNA replication*, the two strands open locally, and each strand becomes the template for synthesizing the opposite strand, its sequence dictated by base complementarity. The DNA double helix is rarely exposed. It is typically wrapped around histone proteins and packaged in a coiled structure referred to as *chromatin*.

The complete DNA content of an organism is referred to as its *genome*, and is contained in one or more large uninterrupted pieces called *chromosomes*. Prokaryotic cells contain one circular chromosome, and eukaryotic cells contain varying numbers of linear chromosomes (16 in yeast, 23 pairs in human) that are compartmentalized within the cell *nucleus*. Each linear chromosome is marked by a well-defined central region, the *centromere* and the chromosomal endpoints called *telomeres*. In a multi-cellular organism, every cell contains an identical copy of the genome (with extremely few exceptions such as red blood cells that do not have a nucleus). In addition to the chromosomal DNA, cells typically contain additional small pieces of DNA in plasmids (small circular pieces found in bacteria and typically containing antibiotic resistance genes), or mitochondria and chloroplasts (energy production organelles found in

eukaryotes). Genome size varies widely across species, typically 5kb-200kb (kilo-bases) for viruses<sup>20-22</sup>, 500kb to 5Mb for bacteria<sup>15</sup>, 10-30Mb for unicellular fungi<sup>23,24</sup>, 97Mb for the worm<sup>25</sup>, 165Mb for the fly<sup>26</sup>, 2-3Gb for mammals<sup>19,27</sup>, and 100Mb-100Gb for plants<sup>28</sup>.

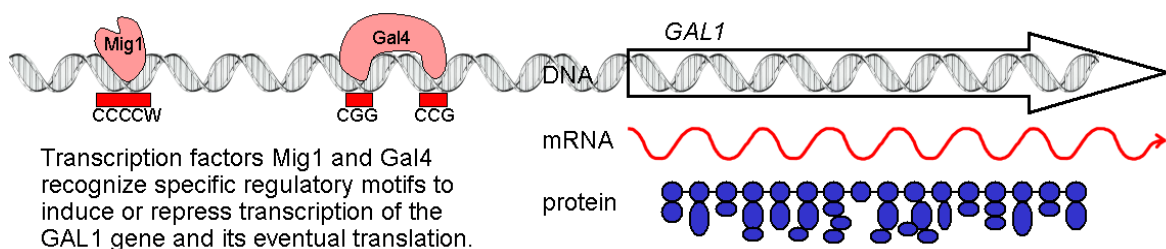
The amino-acid sequence of every protein is encoded within a single continuous stretch of DNA called a *gene*. The transfer of information from the four-letter nucleotide alphabet of DNA to the 20 amino-acid alphabet of proteins is ensured by a process called *translation*. Consecutive nucleotide triplets (*codons*) are translated into consecutive amino-acid residues, according to a precise translation table, referred to as the *genetic code*. There are 64 possible codons and only 20 amino acids, hence the genetic code contains degeneracies, and the same amino acid can be encoded by multiple codons. Additionally, the codon ATG (that codes for Methionine) also serves as a special translation initiation signal, and three codons (TGA, TAG, TAA) are dedicated translation termination signals. These are typically called *start* and *stop* codons. DNA is a *directional molecule*, and so are proteins. DNA is always read and synthesized in the 5' to 3' direction (named after the 5' and 3' carbons in the carbon-ring of the sugar). Given this directionality of either strand, we can refer to sequences *upstream* (5') or *downstream* (3') of a particular nucleotide on the same strand. The two complementary strands run in opposite direction and are called anti-parallel, hence upstream in one strand is complementary to downstream on the opposite strand. Upstream and downstream are typically used in relation to the coding strand of a gene (containing the sequence ATG). Proteins are synthesized from the N terminus (encoded by the 5' part of the gene) to the C terminus (encoded by the 3' part of the gene).

## **0.2. Gene regulation and the dynamic cell**

DNA is not directly translated into protein, but it is first transferred by complementarity into an intermediary single-stranded information carrier called messenger RNA or *mRNA* in a process called *transcription*. The *Central Dogma* of biology refers to this transfer of the genetic information from DNA to RNA to protein. RNA is similar to DNA, but is single-stranded and contains a different type of sugar connector between the phosphate backbone and the variable base (also the four bases are A,C,G,U instead of A,C,G,T). This difference in structure enables RNA to assume complex three-dimensional folds and perform a variety of cellular functions, only one of

which is information transfer between DNA and protein. In eukaryotic cells, transcription occurs in the nucleus where the DNA resides, and the resulting mRNA molecule is then transferred outside the nucleus where the translation machinery resides. During this transfer, the *transcript* undergoes a maturation step, including the excision (called *splicing*) of untranslated gene portions (called *introns*), and the joining of the remaining portions of the transcribed gene that are typically translated (called *exons*). The splicing of introns is dictated by subtle signals between 6 and 8 bp (base pairs) long that are found mainly at the junctions between exons and introns and within each intron. In prokaryotic cells, transcripts do not undergo splicing and sometimes contain multiple consecutively translated genes of related function.

The process of protein and RNA production, also called *gene expression*, is tightly controlled at multiple stages, but mainly at the stage of *transcription initiation*. This involves the uncoiling of chromatin structure around the gene to be expressed and the recruitment of a number of protein players that include the transcription machinery. These processes are regulated by a specific class of DNA-binding proteins called *transcription factors*. These bind the double-stranded DNA helix in sequence-specific *binding sites*, recognizing electrostatic properties of the nucleotides at each contact point. A *regulatory motif* describes the sequence specificity of a transcription factor, namely, the nucleotide patterns that are in common to the sites bound. Transcription factors are classified according to their effect on the expression of their target genes: an *activator* increases the level of gene expression when bound, and a *repressor* decreases that level. Transcription factor binding is modulated by the protein concentration and localization of the transcription factor, the three-dimensional conformation of the transcription factor that may depend on chemical modifications, protein-protein interactions with other factors that may bind cooperatively or competitively, and chromatin accessibility



**Figure 0.1. The Central Dogma of Biology.** DNA makes RNA makes protein

surrounding the binding site. Finally, in addition to transcription initiation, gene expression is regulated at many stages, including mRNA transport and splicing, translation initiation and efficiency, mRNA stability and degradation, post-translational modifications of a protein, and protein stability.

These processes together modulate gene expression in response to environmental changes, and are interlinked in complex *regulatory networks*, responsible for the dynamic nature of the cell. These dynamics create the multitude of specific cell responses to varying environmental stimuli. Gene regulation also creates the incredible variety of cell types found within the same organism. For example heart, liver, lung, nail, skin, eye, neurons, hair, or bone all have the exact same DNA content, but express a different set of genes. Changes in gene expression however, can also be responsible for a number of complex diseases. Understanding the dynamic cell is a major challenge for molecular biology and modern medicine.

### **0.3. Evolutionary change and comparative genomics**

The *evolution* of these complex mechanisms was shaped by the forces of random change and natural selection. Random genomic change can generate new functions or disrupt existing ones, and natural selection favors and keeps the fittest combinations. The *genotypic* differences accumulated at the DNA level lead to observed *phenotypic* differences between individuals of a population. Genomic changes can be as subtle as the mutation, insertion or deletion of individual nucleotides, and as drastic as the duplication or loss of chromosomal segments, entire chromosomes, or complete genomes. Changes in a protein-coding gene can lead to multiple co-existing variants, or *alleles*, of that gene within a population, that differ in specific residues and perform the same function with slight differences. As the result of mating, the progeny will inherit a combination of paternal and maternal alleles for different genes. The random mating of individuals within a populations and the random segregation of chromosomal segments in gamete formation creates new allelic combinations at each generation. The frequency of these allelic combinations will vary through evolutionary time, either by selection for their evolutionary fitness or by random genetic drift. As populations segregate and adapt to their environment, different combinations of alleles dominate in each population. The resulting differences in behavior or chromosomal organization can lead to loss of

reproductive ability across sub-populations and the emergence of new *species*. The emergence of new functions in these changing species allowed adaptation to all niches on land, in the air, underground, or in the deepest oceans, in species as diverse as dinosaurs and amoebae. It is thought that all life in the planet descends from a single ancestral cell that lived around 3.5 billion years ago, and the incredible biodiversity observed today resulted from incremental changes of existing life forms.

The genomes of related species exhibit similarities in functional elements that have undergone little change since the species' common ancestor. Deleterious mutations in these functional regions have certainly occurred, but the individuals carrying them have been at a disadvantage and eventually eliminated by natural selection. Mutations in non-functional regions have no effect to an organism's reproductive fitness, and will accumulate over evolutionary time. Hence, the combined effects of random mutation and natural selection allow comparative approaches to separate conserved functional regions from diverged non-functional regions. Comparative genome analysis of related species should provide a general approach for identifying functional elements without prior knowledge of function, by virtue of having a greater degree of conservation across the genomes of related species. When selecting species for a pairwise comparative analysis, we face a tradeoff between closely related species (with many common functional elements but additional spuriously conserved non-functional regions), and distantly related species (with mostly diverged non-functional regions but fewer common functional elements). The use of multiple closely-related species may present an attractive alternative, exhibiting an accumulation of independent mutations in non-functional regions, while having most biological functions in common.

Recent studies have demonstrated the potential power of comparative genomic comparison. Cross-species conservation has previously been used to identify putative genes or regulatory elements in small genomic regions<sup>10-13</sup>. Light sampling of whole-genome sequence has been studied as a way to improve genome annotation<sup>4,14</sup>. Complete bacterial genomes have been compared to identify pathogenic and other genes<sup>15-18</sup>. Genome-wide comparison has been used to estimate the proportion of the mammalian genome under selection<sup>19</sup>.

#### 0.4. Sequence alignment and phylogenetic trees

The comparison of related sequences is typically represented as *sequence alignment* (for an example see figure 3.2). The correspondence of nucleotides across the sequences compared is given by offsetting the nucleotides of each sequence such that matching nucleotides are stacked at the same index across all sequences. To represent insertions or deletions (*indels*), gaps are typically inserted as dashes in the shorter sequence; these could represent a deletion in the sequence containing the gap, or an insertion in the other sequences. Typically, no reordering or repetition of nucleotides is allowed within a sequence, and hence no inversions, duplications, or translocations are represented in a sequence alignment. To construct an alignment of two sequences is equivalent to finding the optimal path in a two-dimensional grid of cells, and dynamic programming algorithms have been developed to align two sequences in time proportional to the product of their lengths, and space proportional to sum of their lengths. The optimal alignment of two sequences minimizes the total cost of insertions, deletions, and nucleotide substitutions (gaps and mismatches), each penalized according to input parameters. These parameters are set to match estimated rates of insertions, deletions and nucleotide substitutions in well-conserved portions of carefully-constructed alignments. For example, substitutions between nucleotides of similar structure are more frequent and hence *transitions* between *purines* (A and G) or between *pyrimidines* (C and T) are penalized less than *transversions* from a purine to a pyrimidine and vice versa. Also, it is typical to penalize gaps using affine functions, namely adding a cost proportional to the size of the gap to a fixed cost for starting a gap. *Global alignments* compare the entire length of the sequences compared, and *local alignments* only align sub-portions of the sequences.

The best *match* of a *query* sequence can be found in a *database* of sequences by scoring the local alignments between the query and each sequence in the database. Constructing the full dynamic programming matrix for each of the sequences in a large database can be costly, and efficient algorithms have been developed to only align a small subset of the database sequences. These algorithms take advantage of the fact that strong matches of a query sequence will typically contain stretches of perfectly conserved residues, and first select all database sequences that contain such stretches. To do so, a



hash table is first constructed for the database, listing all sequences and positions that contain a particular k-mer. After this slow step that need only be performed once, the lookup of all k-mers in a query sequence can be performed rapidly against a large database, constructing a list of *hits*. Local alignments are then constructed around each hit, extending the k-mer matches to longer high-scoring local alignments. These ideas are implemented in the popular program BLAST, and used thousands of times daily to query the genomes of dozens of sequenced species and millions of sequences. One modification of the BLAST algorithm called two-hit Blast only constructs a local alignment when at least two nearby hits are found. This allows the retrieval of more distantly related sequences by searching for shorter k-mers, while still maintaining high specificity by requiring multiple k-mer hits in common.

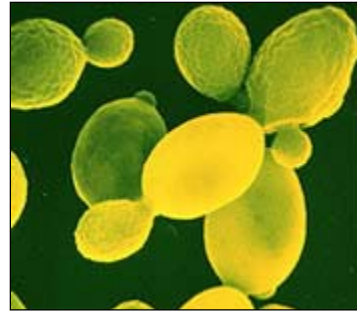
*Multiple sequence alignments* can also be constructed for more than two sequences. Constructing the full dynamic programming matrix is exponential in the number of sequences compared and typically impractical for long sequences. Therefore, current algorithms work by extending multiple pairwise alignments between the sequences compared. The similarities between all pairs of sequences can be used to construct a *phylogenetic tree*, summarizing the most likely ancestry of the sequences, linking them hierarchically from the most closely related pair to the most distantly related outgroup. Multiple sequence alignment algorithms typically start by aligning the most closely related sequences, and progressively merge alignments moving up the phylogenetic tree from the leaves to the root. Algorithms to merge two alignments typically use once-a-gap-always-a-gap methods, but more recent algorithms have been developed to locally re-optimize multiple alignment portions by revisiting previously added gaps and improving the overall alignment score.

### **0.5. Model organisms and yeast genetics.**

The shared biology of related species allows one to study a biological process in one organism and apply the knowledge to another organism. Simpler organisms provide excellent models for developing and testing the procedures needed for studying the much more complex human genome. Such *model organisms* include bacteria, yeast, fungi, worms, flies and mice, each teaching us different aspects of human biology. For example, the study of cancer development has flourished by studying mouse models, and

has lead to medical application in humans. *Mutant* strains can be isolated containing specific defects in genes that lead to disease phenotypes. Controlled *crosses* can be used to restore lost functions or inhibit genes at particular stages of development and study their effects on the organism. The shorter the generation time of a model organism, the easier it is to perform multiple crosses.

The yeast *Saccharomyces cerevisiae* in particular provides a powerful genetic system with the availability of a wide array of tools such as gene replacement, plasmids, deletion strains, two-hybrid systems. Yeast is also amenable to biochemical methods, such as the purification and characterization of protein complexes. Because of these experimental advantages, yeast has been the system of choice to study the most basic cellular functions common to eukaryotes such as cell division, cell structure, energy production, cell growth, cell death, cell cycle, gene regulation, transcription initiation, cell signaling, and other basic cell processes. More recently, yeast has become the organism of choice for the development and testing of modern technologies for *genome-wide* experimental studies. The complete parts-list of all genes has radically changed the face of biological research. If a particular phenotype is due to the function of a single protein, it is necessarily encoded by one of these few thousand genes. Additionally, the relatively small number of genes (~6000) allows the simultaneous observation of the complete genome for mRNA expression, transcription factor binding, or protein-protein interactions. The public sharing of yeast strains, materials, and genome-wide experimental data has provided a global view of the dynamic yeast genome unmatched in any other organism.



**Figure 0.2.** The yeast *Saccharomyces cerevisiae* undergoing cell division.

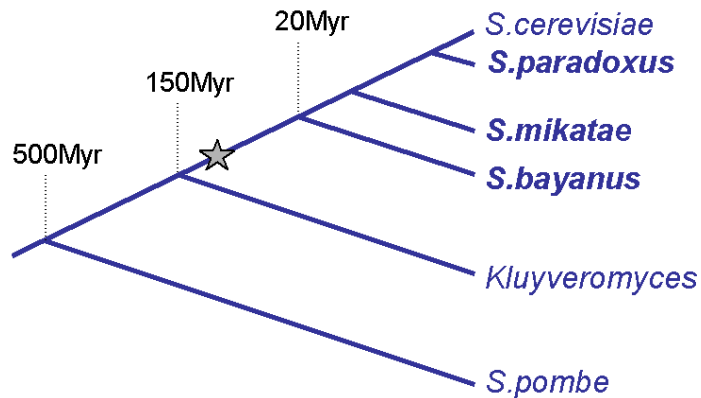
Yeast also presents an ideal organism for developing computational methods for genome-wide *comparative* analysis. It is the most well-studied eukaryote, and the vast functional knowledge allows the immediate validation of our findings against previous work. Additionally, the strong experimental system allows the experimental follow-up of biological hypotheses raised in the comparative work. The small genome size (250 times

smaller than human) allows the sequencing of multiple yeast species at an affordable cost. Additionally, the small number of repetitive elements allows for easy whole-genome-shotgun assembly (see next section). For all these considerations, we decided to work on yeast.

## 0.6. Genome sequencing and assembly

We sequenced and assembled the complete genomes of *S. paradoxus*, *S. mikatae* and *S. bayanus*, three yeast species that are close relatives of *S. cerevisiae*, within the *Saccharomyces sensu stricto* group<sup>29</sup>. Their divergence times from the *S. cerevisiae* lineage are approximately 5, 10 and 20 million years (based on sequence divergence of ribosomal DNA sequence).

Like *S. cerevisiae*, they all have 16 chromosomes and their genomes contain about 12 million bases. These species were chosen based on their evolutionary relationships (closely enough related that functional elements be conserved, and distant enough that non-functional bases have had enough evolutionary time to diverge).



**Figure 0.3: Phylogenetic tree of analyzed species.** The newly sequenced species are shown in bold. Star denotes inferred genome-wide duplication of the yeast genome. Divergence times are approximate and based on ribosomal DNA sequence divergence

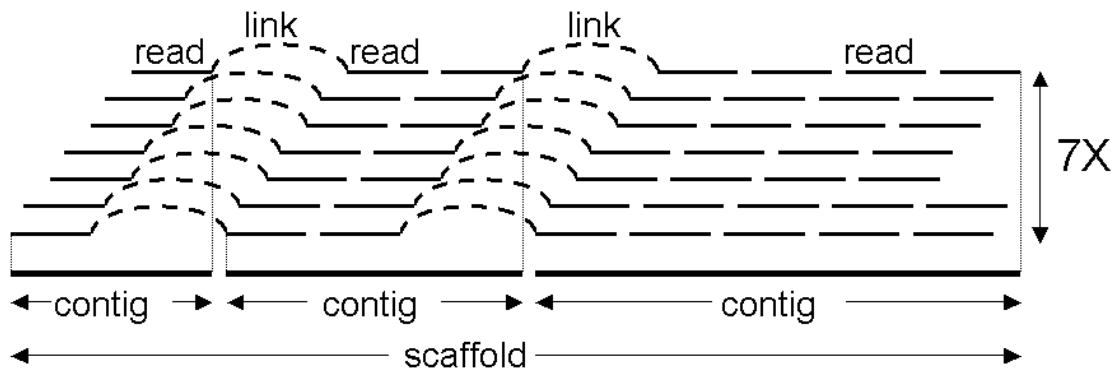
Reading the order of the nucleotides in any one segment of DNA relies on a technology developed by Sanger in 1977 that uses the central agent of DNA replication, *DNA polymerase*. This protein complex recognizes the transition from double-stranded DNA to single-stranded DNA in an incomplete helix, and extends the shorter strand in the 5' to 3' direction. By introducing a small fraction of faulty nucleotides that cause an early termination of the extension reaction, and subsequently comparing the lengths of resulting fragments in each of four reactions, this method infers the sequence of a DNA fragment. The extension reaction can be initiated at any unique segment of DNA by

introducing a complementary segment called a *primer*. This primer binds single-stranded DNA by complementarity, creating the double-strand to single-strand transition recognized by DNA polymerase. Unfortunately, since the Sanger method works by weight separation between fragments of different lengths, it can only determine the sequence of small fragments (currently around 800 nucleotides). The weight difference between fragments of 800 nucleotides and fragments of 801 nucleotides is too small to be detected reliably.

To obtain the sequence of longer stretches of DNA, two methods are possible. One is to synthesize a new primer at the end of 800 nucleotides and use it to sequence the subsequent 800 nucleotides (and so on). Unfortunately, synthesizing new primers is expensive and time-consuming since the primer to be used is not known until the sequence is obtained, and this method is rarely used. An alternative method is to first make many copies of the longer stretch of DNA and randomly break them into small fragments, and then sequence 800 nucleotide *reads* from each of these fragments and re-piece them together computationally (each of the fragments is inserted to a common *vector* whose sequence is known, hence the same primer can be used to sequence the end of each of these fragments). This alternative method is called *shotgun sequencing*, in reference to the random breaking of the longer fragment as if struck by a shotgun. Sequence reads can also be obtained from both ends of a fragment, providing *linking* information between *paired reads*. This method is called *paired-end shotgun sequencing*. The shotgun fragments are typically selected to be of a particular size, providing additional information about the genomic distance between paired sequence reads.

Shotgun sequencing depends heavily on the computational ability to correctly *assemble* the resulting fragments of sequence. *Fragment assembly* searches for sequences common between two sequence fragments (also called *reads*) and unique otherwise, in order to join them into a longer sequence. This is made harder due to sequencing errors that lead to sequence differences between reads that really come from the same part of the genome, as well as repetitive sequences within genomes that lead to identical sequences between reads that come from different parts of the genome. Modern assembly programs produce stretches of continuous sequence called *contigs*, which are

linked into *supercontigs* or *scaffolds*, when their relative order, orientation, and estimated spacing is given by the pairing of reads (Figure 0.4). To assemble complete genomes, two methods are currently in use. *Whole-genome shotgun* (WGS) randomly breaks the complete genome and assembles all fragments computationally. *Clone-based* methods first partition the genome into large fragments (clones) and then use shotgun sequencing for each of the fragments. Clone-based methods are more expensive but more reliable. WGS methods are cheaper but rely more heavily on the ability of subsequent computational assembly programs. Hybrids between WGS and clone-based methods are used nowadays in major sequencing projects. It is also common to use WGS with links of multiple sizes to provide both short-range and long-range connectivity information.



**Figure 0.4 Genome Assembly.** Overlapping sequence reads are grouped into blocks of continuous sequence (contigs). The pairing of forward and reverse reads provides links across neighboring contigs, grouping them in supercontigs or scaffolds. Each base in the genome is observed on average in 7 overlapping reads.

## CHAPTER 1: GENOME CORRESPONDENCE

### 1.1. Introduction

The first issue in comparative genomics is determining the correct correspondence of chromosomal segments and functional elements across the species compared. This involves the recognition of *orthologous* segments of DNA that descend from the same region in the common ancestor of the species compared. However, it is equally important to recognize which segments have undergone duplication events, and which segments were lost since the divergence of the species. By accounting for duplication and loss events, we ensure that we are comparing orthologous segments.

We decided to use genes as discrete genomic anchors in order to align and compare the species. We constructed a bipartite graph connecting annotated protein-coding genes in *S. cerevisiae* to predicted protein-coding genes in each of the other species based on sequence similarity at the amino-acid level. This bipartite graph should contain the orthologous matches but also contains spurious matches due to shared domains between proteins of similar functions, and gene duplication events that precede the divergence of the species. Determining which matches represent true orthologs and resolving the correspondence of genes across the four species will be the topic of this chapter.

We present an algorithm for comparative annotation that has a number of attractive features. It uses a simple and intuitive graph theoretic framework that makes it easy to incorporate additional heuristics or knowledge about the genes at hand. It represents matches between sets of genes instead of only one-to-one matches, thus dealing with duplication and loss events in a very straightforward way. It uses the chromosomal positions of the compared genes to detect stretches of conserved gene order and uses these to resolve additional orthologous matches. It accounts for all genes compared, resolving *unambiguous* matches instead of simply *best* matches, thus ensuring that all 1-to-1 genes are true orthologs. It works at a wide range of evolutionary distances, and can cope with unfinished genomes containing gaps even within genes.

## 1.2. Establishing gene correspondence

Previously described algorithms for comparing gene sets have been widely used for various purposes, but they are not applicable to the problem at hand.

Best Bidirectional Hits (BBH)<sup>30,31</sup> looks for gene pairs that are best matches of each other and marks them as orthologs. In the case of a recent gene duplication however, only one of the duplicated genes will be marked as the ortholog without signaling the presence of additional homologs. Thus, no guarantees are given that 1-to-1 matches will represent orthologous relations and incorrect matches may be established.

Clusters of Orthologous Genes (COG)<sup>32,33</sup> goes a step further and matches groups of genes to groups of genes. Unfortunately, the grouping is too coarse, and clusters of orthologous genes typically correspond to gene families that may have expanded before the divergence of the species compared. This inability to distinguish recent duplication events from more ancient duplication events makes it inapplicable in this case, since the genome of *S. cerevisiae* contains hundreds of gene pairs that were anciently duplicated before the divergence of the species at hand<sup>34</sup>. COGs would not distinguish between the two copies of anciently duplicated genes, and many orthologous matches would not be detected (Koonin, personal communication).

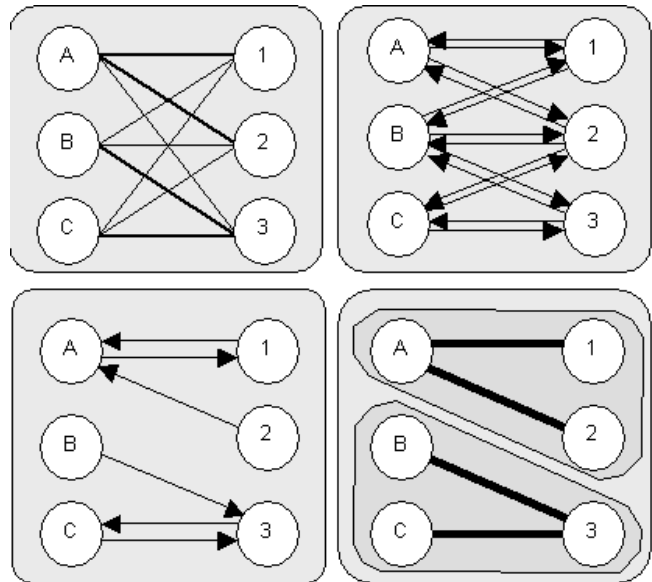
We introduce the concept of a Best Unambiguous Subset (BUS), namely a group of genes such that all best matches of any gene within the set are contained within the set, and no best match of a gene outside the set is contained within the set. A BUS builds on both BBHs and COGs to resolve the correspondence of genes across the species. The algorithm, at its core, represents the best match of every gene as a set of genes instead of a single best hit, which makes it more robust to slight differences in sequence similarity. A BUS can be isolated from the remainder of the bipartite gene correspondence graph while preserving all potentially orthologous matches. BUS also allows a recursive application grouping the genes into progressively smaller subsets and retaining ambiguities until later in the pipeline when more information becomes available. Such information includes the conserved gene order (synteny) between consecutive orthologous genes that allows the resolving of additional neighboring genes.

### 1.3. Overview of the algorithm

We formulated the problem of genome-wide gene correspondence in a graph-theoretic framework. We represented the similarities between the genes as a bipartite graph connecting genes between two species. We weighted every edge connecting two genes by the amino acid sequence similarity between the two genes, and the overall length of the match.

We separated this graph into progressively smaller subgraphs until the only remaining matches connected true orthologs (Figure 1.1). To achieve this separation, we eliminated edges that are sub-optimal in a series of steps. As a pre-processing step, we eliminated all edges that are less than 80% of the maximum-weight edge both in amino acid identity and in length. Based on the unambiguous matches that resulted from this step, we built blocks of conserved gene order (synteny) when neighboring genes in one species had one-to-one matches to neighboring genes in the other species; we used these blocks of conserved synteny to resolve additional ambiguities by preferentially keeping matches within synteny blocks. We finally searched for subsets of genes that are locally optimal, such that all best matches of genes within the group are contained within the group, and no genes outside the group have matches within the group. These best unambiguous subsets (BUS) ensure that the bipartite graph is maximally separable, while maintaining all possibly orthologous relationships.

When no further separation was possible, we returned the connected components of the final graph. These contain the one-to-one orthologous pairs resolved as well as sets of genes whose correspondence remained ambiguous in a small number of homology groups.



**Figure 1.1. Overview of graph separation.**

We construct a bipartite graph based on the blast hits. We consider both forward and reverse matches for near-optimality based on synteny and sequence similarity. Sub-optimal matches are progressively eliminated simplifying the graph. We return the connected components of the undirected simplified graph.

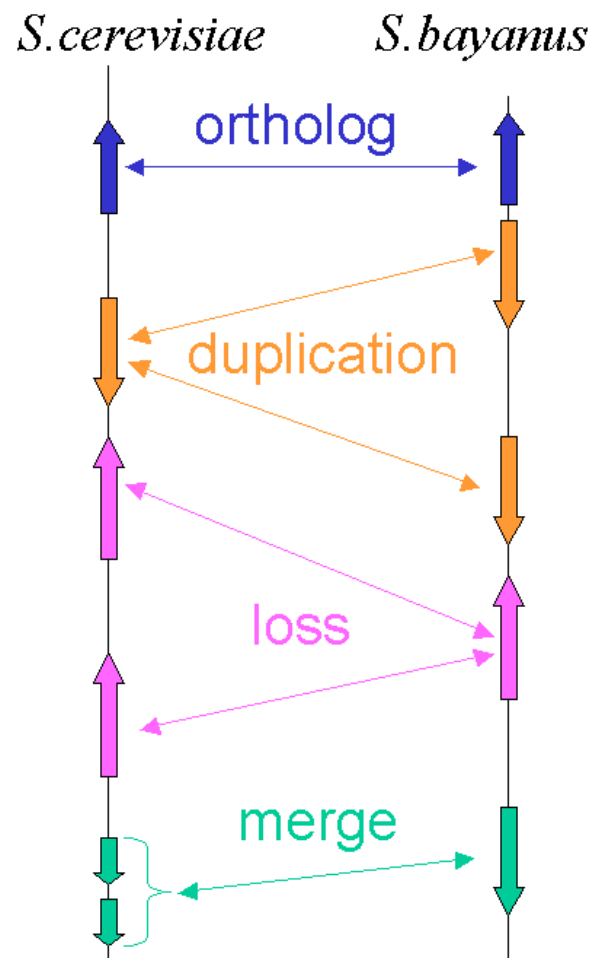


#### 1.4. Automatic annotation and graph construction

In this section, we describe the construction of the weighted bipartite graph  $G$ , representing the gene correspondence across the species compared. We started with the genomic sequence of the species and the annotation of *S. cerevisiae*, namely the start and stop coordinates of genes. We then predicted protein-coding genes for each newly sequenced genome. Finally we connected across each pair of species the genes that shared amino-acid sequence similarity.

The input to the algorithm is based on the complete genome for each species compared. For *S. cerevisiae*, we used the public sequence available from the Saccharomyces Genome Database (SGD) at [genome-www.stanford.edu/Saccharomyces](http://genome-www.stanford.edu/Saccharomyces). SGD posts sixteen uninterrupted sequences, one for each chromosome. The sequence was obtained by an international sequencing consortium and published in 1996. It was completed by a clone-based sequencing approach and directed sequence finishing to close all gaps. Subsequent to the publication, updates to the original sequence have been incorporated in SGD based on resequencing of regions studied in labs around the world.

The genome sequence of *S. paradoxus*, *S. mikatae* and *S. bayanus* was obtained at the MIT/Whitehead Institute Center for Genome Research. We used a whole-genome shotgun sequencing approach with paired-end sequence reads of 4kb plasmid clones, with lab protocols as described at [www-genome.wi.mit.edu](http://www-genome.wi.mit.edu). We used ~7-fold



**Figure 1.2. Bipartite Graph Construction.**

Annotated ORFs (vertical block arrows) are connected based on sequence similarity.

redundant coverage, namely every nucleotide in the genome was contained on average in at least 7 different reads. The information was then assembled with the Arachne computer program<sup>35,36</sup> into a draft sequence for each genome. The assembly contains *contigs*, namely continuous blocks of uninterrupted sequence, and *scaffolds* or *supercontigs*, namely uninterrupted blocks of linked contigs for which the relative order and orientation is known. This order and orientation is given by the pairing of reads that originated from the ends of the same 4kb clone. The draft genome sequence of each species has long-range continuity (more than half of the nucleotides are in scaffolds of length 230-500 kb, as compared to 942 kb for the finished sequence of *S. cerevisiae*), relatively short sequence gaps (0.6-0.8 kb, which is small compared to a typical gene), and contains the vast majority of the genome (~95%).

Once the genome sequences are available, we determine the set of protein-coding genes for each species. For *S. cerevisiae*, we used the public gene catalogue at SGD. It was constructed by including all predicted protein coding genes of at least 100 AA that do not overlap longer genes by more than 50% of their length. It was subsequently updated to include additional short genes supported by experimental evidence and to reflect changes in the underlying sequence when resequencing revealed errors. For the three newly sequenced species, we predicted all uninterrupted genes starting with a methionine (start codon ATG) and containing at least 50 amino acids.

We then constructed the bipartite graph connecting all predicted protein coding genes that share amino acid sequence similarities across any two species (Figure 1.2). For this purpose, we first used protein BLAST<sup>37</sup> to find all protein hits between the two protein sets (we used WU-BLAST BlastP with parameters W=4 for the hit size in amino acids, hitdist=60 for the distance between two hits and  $E=10^{-9}$  for the significance of the matches reported). Since the similarity between query protein  $x$  in one genome and subject protein  $y$  in another genome is sometimes split in multiple blast hits, we grouped all blast hits between  $x$  and  $y$  into a single *match*, weighted by the average amino acid percent identity across all hits between  $x$  and  $y$  and by the total protein length aligned in blast hits. These matches form the edges of the bipartite graph  $G$ , described in the following section.

### 1.5. Initial pruning of sub-optimal matches

Let  $\mathbf{G}$  be a weighted bipartite graph describing the similarities between two sets of genes  $\mathbf{X}$  and  $\mathbf{Y}$  in the two species compared (Figure 1.1, top left panel). Every edge  $e=(x,y)$  in  $E$  that connects nodes  $x \in X$  and  $y \in Y$  was weighted by the total number of amino acid similarities in BLAST hits between genes  $x$  and  $y$ . When multiple BLAST hits connected  $x$  to  $y$ , we summed the non-overlapping portions of these hits to obtain the total weight of the corresponding edge. We constructed graph  $\mathbf{M}$  as the directed version of  $\mathbf{G}$  by replacing every undirected edge  $e=(x,y)$  by two directed edges  $(x,y)$  and  $(y,x)$  with the same weight as  $e$  in the undirected graph (Figure 1.1, top right panel). This allowed us to rank edges incident from a node, and construct subsets of  $\mathbf{M}$  that contain only the top matches out of every node.

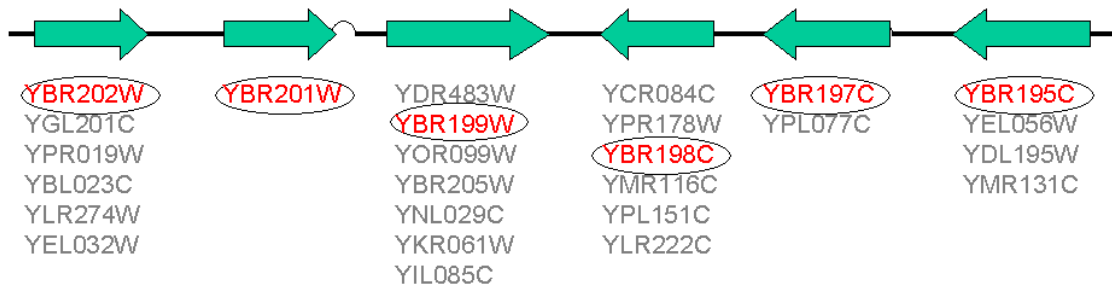
This step drastically reduced the overall graph connectivity by simply eliminating all out-edges that are not near optimal for the node they are incident from. We defined  $\mathbf{M}_{80}$  as the subset of  $\mathbf{M}$  containing for every node only the outgoing edges that are at least 80% of the best outgoing edge (any not in the upper 20% of all scores). This was mainly a preprocessing step that eliminated matches that were clearly non-optimal. Virtually all matches eliminated at this stage were due to protein domain similarity between distantly related proteins of the same super-family or proteins of similar function but whose separation well-precedes the divergence of the species. Selecting a match threshold relative to the best edge ensured that the algorithm performs at a range of evolutionary distances. After each stage, we separated the resulting subgraph into connected components of the undirected graph (Figure 1.1, bottom right panel).

### 1.6. Blocks of conserved synteny

The initial pruning step created numerous two-cycle subgraphs (unambiguous one-to-one matches) between proteins that do not have closely related paralogs. We used these to construct blocks of conserved synteny based on the physical distance between consecutive matched genes, and preferentially kept edges that connect additional genes within the block of conserved gene order (Figure 1.3). Edges connecting these genes to genes outside the blocks were then ignored, as unlikely to represent orthologous relationships. Without imposing an ordering on the scaffolds or the chromosomes, we

associated every gene  $x$  with a fixed position ( $s$ , start) within the assembly, and every gene  $y$  with a fixed position (chromosome, start) within *S. cerevisiae*. If two one-to-one unambiguous matches  $(x_1, y_1)$  and  $(x_2, y_2)$  were such that  $x_1$  was physically near  $x_2$ , and  $y_1$  was physically near  $y_2$ , we constructed a synteny block  $B = (\{x_1, x_2\}, \{y_1, y_2\})$ . Thereafter, for a gene  $x_3$  that was proximal to  $\{x_1, x_2\}$ , if an outgoing edge  $(x_3, y_3)$  existed such that  $y_3$  was proximal to  $\{y_1, y_2\}$ , we ignored other outgoing edges  $(x_3, y')$  if  $y'$  was not proximal to  $\{y_1, y_2\}$ .

Without this step, duplicated genes in the yeast species compared remained in two-by-two homology groups, especially for the large number of ribosomal genes that are nearly identical to one another. We found this step to play a greater role as evolutionary distances between the species compared became larger, and sequence similarity was no longer sufficient to resolve all the ambiguities. We only considered synteny blocks that had a minimum of three genes before using them for resolving ambiguities, to prevent being misled by rearrangements of isolated genes. We set the maximum distance  $d$  for considering two neighboring genes as proximal to 20kb, which corresponds to roughly 10 genes. This parameter should match the estimated density of syntenic anchors. If many genomic rearrangements have occurred since the separation of the species, or if the scaffolds of the assembly are short, the syntenic segments will be shorter and setting  $d$  to larger values might hurt the performance. On the other hand if the number of unambiguous genes is too small at the beginning of this step, the genes used as anchors will be sparse, and no synteny blocks will be possible for small values of  $d$ .



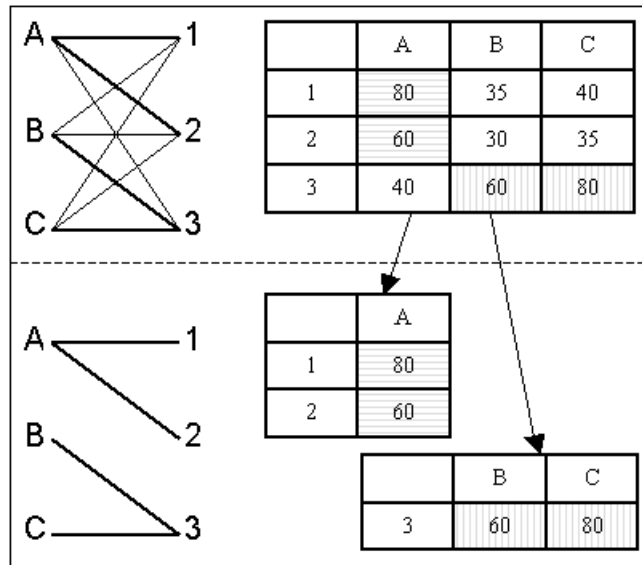
**Figure 1.3. The use of synteny.** In blocks of conserved gene order (synteny), we preferentially keep those matches that preserve the order of orthologous genes.

## 1.7. Best Unambiguous Subsets

We finally separated out subgraphs that were connected to the remaining edges in the graph by solely non-maximal edges. These subgraphs are such that the best match of any node within the subset is contained within the subset, and no node outside the subset has its best match within the subset. These two properties ensure that the subsets are both best and unambiguous.

We defined a Best Unambiguous Subset (BUS) of the nodes of  $X \cup S$ , to be a subset  $S$  of genes, such that  $\forall x: x \in S \Leftrightarrow \text{best}(x) \subseteq S$ , where  $\text{best}(x)$  are the nodes incident to the maximum weight edges from  $x$ . We then constructed  $M_{100}$ , following the notation above, namely the subset of  $M$  that contains only best matches out of a node. Note that multiple best matches were possible based on our definition. To construct a BUS, we started with the subset of nodes in any cycle in  $M_{100}$ . We augmented the subset by following forward and reverse best edges, that is including additional nodes if their best match was within the subset, or if they were the best match of a node in the subset. This ensured that separating a subset did not leave any node orphan, and did not remove the strictly best match of any node. When no additional nodes needed to be added, the BUS condition was met.

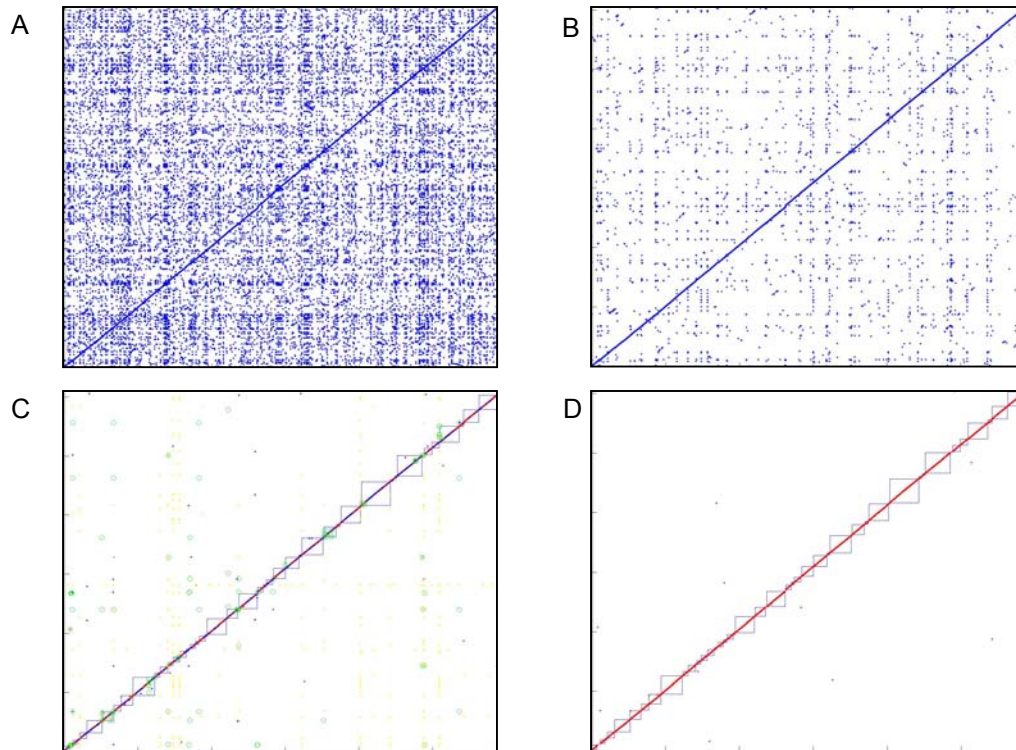
Figure 1.4 shows a toy example of a similarity matrix. Genes A, B, and C in one genome are connected in a complete bipartite graph to genes 1, 2 and 3 in another genome (ignoring for now synteny information). The sequence similarity between each pair is given in the matrix, and corresponds to the edge weight connecting the two genes in the bipartite graph. The set (A,1,2) forms a BUS, since the best matches of A, 1, and 2 are all within the set,



**Figure 1.4. Best Unambiguous Subsets (BUS).** A BUS is a set of genes that can be isolated from a homology group while preserving all potentially orthologous matches. Given the similarity matrix above and no synteny information, two such sets are (A,1,2) and (B,C,3).

and none of them represents the best match of a gene outside the set. Hence, the edges connecting (A,1,2) can be isolated as a subgraph without removing any orthologous relationships, and edges (B,1), (B,2), (C,1), (C,2), (A,3) can be ignored as non-orthologous. Similarly (B,C,3) forms a BUS. The resulting bipartite graph is shown. A BUS can be alternatively defined as a connected component of the undirected version of  $M_{100}$  (Figure 1.1, bottom panels).

This part of the algorithm allowed us to resolve the remaining orthologs, mostly due to subtelomeric gene family expansions, small duplications, and other genes that did not benefit from synteny information. In genomes with many rearrangements, or assemblies with low sequence coverage, which do not allow long-range synteny to be established, this part of the algorithm will play a crucial role.



**Figure 1.5. Performance of the algorithm.** Dotplot representation of the bipartite graph. The 16 chromosomes of *S. cerevisiae* are stacked end-to-end along the y-axis, and the scaffolds of *S. paradoxus* are shown along the x-axis. Every point (x,y) represents an edge between *S. paradoxus* gene y and *S. cerevisiae* gene x. A. Initial bipartite graph. B. Graph resulting from initial disambiguation step. C. Graph resulting from use of BUS and synteny information. D. Unambiguous matches in graph C.

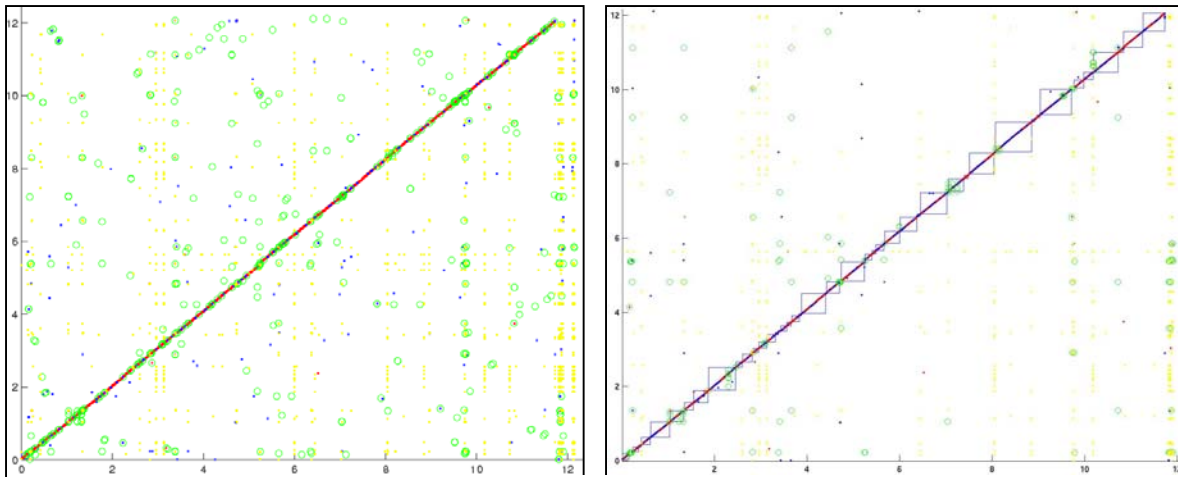
## 1.8. Performance of the algorithm

We applied this algorithm to automatically annotate the assemblies of the three species of yeast. Our Python implementation terminated within minutes for any of the pairwise comparisons. We successfully resolved the graph of sequence similarities between the four species, and found important biological implications in the resulting graph structure.

Figure 1.5 illustrates the performance of the algorithm for the 6235 annotated ORFs in *S. cerevisiae* and all predicted ORFs in *S. paradoxus*. The graph is initially very dense (panel A), the vast majority of edges representing non-orthologous matches, mostly due to protein domain similarities, ancient duplications that precede the time of the common ancestor of the species compared, and transposable elements. After applying the initial pruning step, many of the spurious matches are eliminated (panel B). The presence of unambiguous matches allows us to build blocks of conserved gene order, and use these to resolve additional matches using the BUS algorithm (panel C). The unambiguous 1-to-1 matches are mostly syntenic for *S. paradoxus*, thus ensuring that we are comparing orthologous regions.

More than 90% of genes have clear one-to-one orthologous matches in each species, providing a dense set of landmarks (average spacing ~2 kb) to define blocks of conserved synteny covering essentially the entire genome. Not surprisingly, transposon proteins formed the largest homology groups. The remaining matches were isolated in small subgraphs. These contain expanding gene families that are often found in rapidly recombining regions near the telomeres, and genes involved in environmental adaptation, such as sugar transport and cell surface adhesion<sup>29</sup>. For additional details see section 6.2.

We have additionally experimented running only BUS without the original pruning and synteny steps. More than 80% of ambiguities were resolved, and the remaining matches corresponded to duplicated ribosomal proteins and other gene pairs that are virtually unchanged since their duplication. The algorithm was slower, due to the large initial connectivity of the graph, but a large overall separation was obtained. Figure 1.6 compares the dotplot of *S. paradoxus* and *S. cerevisiae* with and without the use of synteny. Every point represents a match, the x coordinate denoting the position in the *S. paradoxus* assembly, and the y coordinate denoting the position in the *S. cerevisiae*



**Figure 1.6. The effect of using synteny.** Blocks of conserved gene order (blue squares) help resolve additional ambiguities. These are mostly due to pairs of anciently duplicated yeast genes.

genome, with all chromosomes put end-to-end. Lighter dots represent homology containing more than 15 genes (typically transposable elements) and circles represent smaller homology groups (rapidly changing protein families that are often found near the telomeres). The darker dots represent unambiguous 1-to-1 matches, and the boxes represent synteny blocks.

This algorithm has also been applied to species at much larger evolutionary distances, with very successful results (Kellis and Lander, manuscript in preparation). Despite hundreds of rearrangements and duplicated genes separating *S.cerevisiae* and *K.yarrowii*, it successfully uncovered the correct gene correspondence between the two species that are more than 100 million years apart.

Additionally, the algorithm works well with unfinished genomes. By working with sets of genes instead of one-to-one matches, this algorithm correctly groups in a single orthologous set all portions of genes that are interrupted by sequence gaps and split in two or multiple contigs. A best bi-directional hit would match only the longest portion and leave part of a gene unmatched. Finally, since synteny blocks are only built on one-to-one unambiguous matches, the algorithm is robust to sequence contamination. A contaminating contig will have no unambiguous matches (since all features will also be present in genuine contigs from the species), and hence will never be used to build a synteny block. This has allowed the true orthologs to be determined and the contaminating sequences to be marked as paralogs.



This algorithm provides a good solution to determining genome correspondence, works well at a range of evolutionary distances, and is robust to sequencing artifacts of unfinished genomes.

### **1.9. Conclusion.**

We have unambiguously resolved the one-to-one correspondence of more than 90% of *S. cerevisiae* genes. This provides us with a unique dataset whereby we can align and compare the evolutionary pressure of nearly every region in the complete yeast genome across four closely related relatives. In presence of gene duplication, some of the evolutionary constraints that a region is under are relieved, and uniform models of evolution would not capture the underlying selection for these sites. By ensuring that the regions compared are orthologous, we can make uniform assumptions about the rate of change of different regions, and apply statistical models for the significance of strong or weak conservation.

In this thesis, we will use the multiple alignments of the four species to discover protein-coding genes based on the pressure to conserve the reading frame of the amino acid translation (Chapter 2). We will also search for unusually strong conservation in non-coding regions to discover recurring patterns that constitute regulatory motifs (Chapter 3). We will assign functions to these motifs (Chapter 4) and discover their combinatorial interactions (Chapter 5) based on their conserved instances. Finally, we will focus on the differences between the species to discover regions and mechanisms of rapid evolutionary change (Chapter 6).

## CHAPTER 2: GENE IDENTIFICATION

### 2.1. Introduction

The genome of a species encodes genes and other functional elements, interspersed with non-functional nucleotides in a single uninterrupted string of DNA. Recognizing protein-coding genes relies on finding stretches of nucleotides free of stop codons (called Open Reading Frames, or ORFs) that are too long to have likely occurred by chance. Since stop codons occur at a frequency of roughly 1 in 20 in random sequence, ORFs of at least 60 amino acids will occur frequently by chance (5% under a simple Poisson model) and even ORFs of 150 amino acids will appear by chance in a large genome (0.05%). This poses a huge challenge for higher eukaryotes in which genes are typically broken into many, small exons (on average 125 nucleotides long for internal exons in mammals<sup>27</sup>).

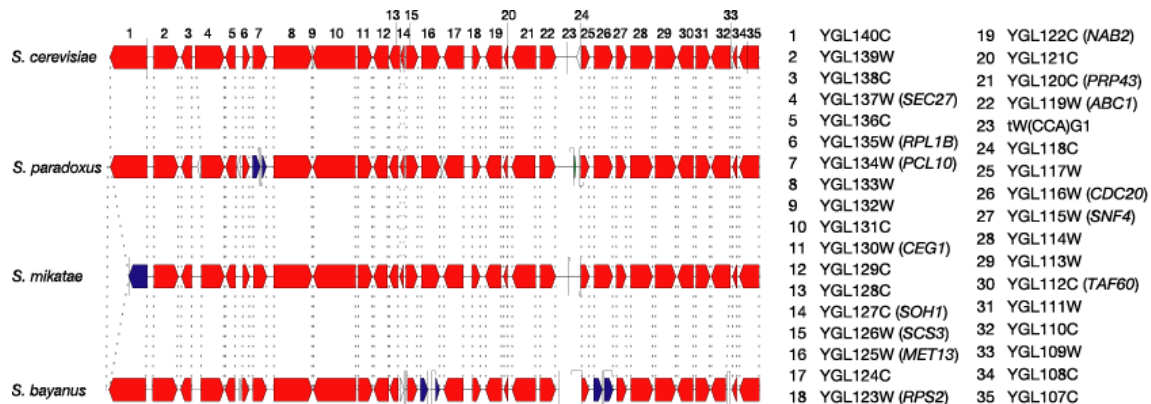
The basic problem is distinguishing *real genes* – those ORFs encoding a translated protein product – from *spurious ORFs* – the remaining ORFs whose presence is simply due to chance. The current public catalogue of yeast genes lists 6062 predicted ORFs that could theoretically encode proteins of at least 100 amino acids. Only two-thirds of these have been experimentally validated (*known*), and the remaining ~2000 ORFs are currently annotated as *hypothetical*. The total number of real protein-coding genes has been a subject of considerable debate, with estimates ranging from 4,800 to 6,400 genes (in mammalian genomes, estimates have ranged from 28,000 to more than 120,000 genes).

In this chapter, we use the comparative information to recognize real genes based on their conservation across evolutionary time. With the availability of genome-wide alignments across the four species, we first examined the different ways by which sequences change in known genes and in intergenic regions. The alignments of known genes revealed a clear pressure to preserve protein-coding potential. We constructed a computational test for reading frame conservation (RFC) and used it to revisit the annotation of yeast. We showed that more than 500 previously annotated ORFs are not meaningful and discovered 43 novel ORFs that were previously overlooked. We additionally refined the gene structure of hundreds of genes, including translation start,

stop, and exon boundaries. We show that our method has high sensitivity and specificity, and suggest changes that affect nearly 15% of yeast genes.

## 2.2. Different conservation of genes and intergenic regions

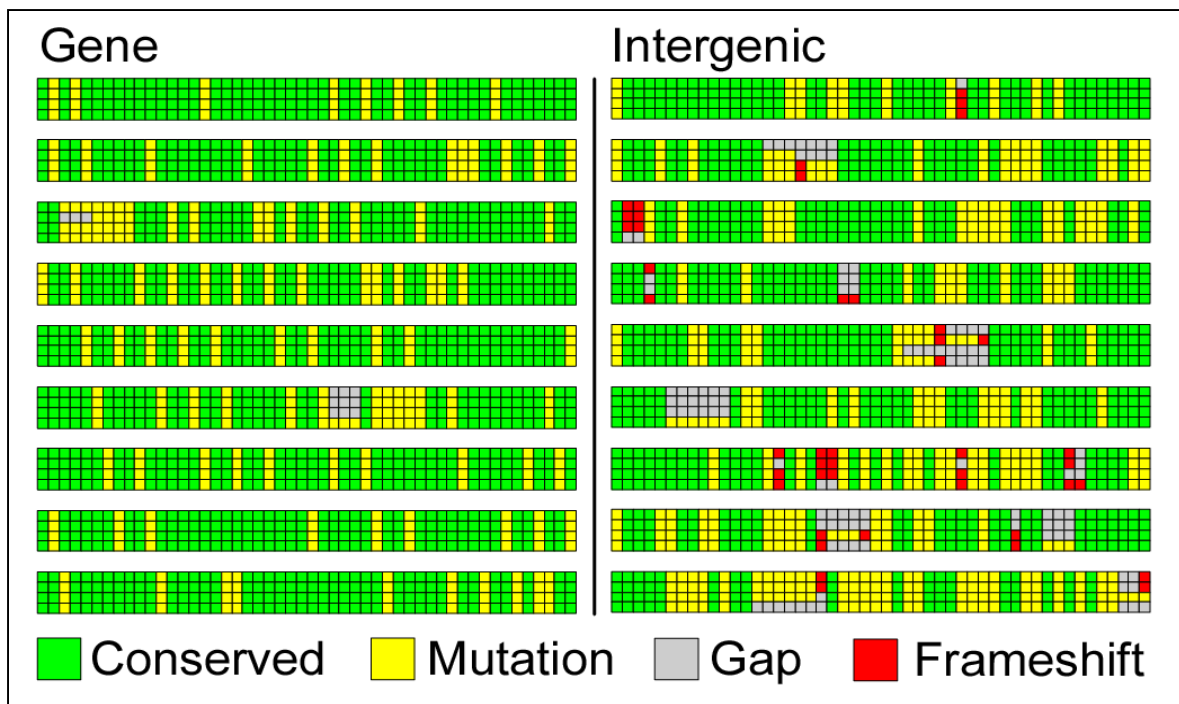
We examined the different types of conservation in genes and intergenic regions. We used the 1-to-1 orthologous anchors (see Chapter 1) to construct a nucleotide-level alignment of the genomes. The strong conservation of local gene order and spacing (Figure 2.1) allowed us to construct genome-wide multiple alignments. We aligned each gene together with its flanking intergenic regions using CLUSTALW<sup>38</sup> for the multiple alignments across the four species. When sequence gaps were present in one or more species, we constructed the alignment in multiple steps. We first aligned the gapless species creating a base alignment. Then we aligned each portion of a partially covered ortholog onto the base alignment, and constructed a consensus for each species based on the individually aligned portions. We marked missing sequence between contigs by a dot and disagreeing overlapping contigs by N. Finally, we constructed a multiple alignment of the four species by merging the piece-wise alignments. With sequence alignments at millions of positions across the four species, it is possible to obtain a precise estimate of the rate of evolutionary change, including substitutions and insertion-deletions (indels), in



**Figure 2.1. Strong conservation of local gene order and spacing** allows genome-wide multiple alignments. A 50kb segment of *S. cerevisiae* chromosome VII aligned with orthologous contigs from each of the other three species. Predicted ORFs are shown as arrows pointing in the direction of transcription. Orthologous ORFs are connected by dotted lines, and colored by the type of correspondence: red for 1-to-1 matches, blue for 1-to-2 matches and white for unmatched ORFs. Sequence gaps are indicated by vertical lines at ends of contigs, with estimated size of gap shown by the length of the hook.

the tree connecting the species. We counted transitions, transversions, insertions and deletions within these alignments and used these to estimate the rate of evolutionary change between the species. We counted the rate of synonymous and non-synonymous substitutions for every protein coding gene to find evidence of positive selection. The detailed results will be described in chapter 6.

We compared the rate of sequence change at aligned sites across the four species in intergenic and genic (protein-coding) regions (Figure 2.2). We found radically different types of conservation. Intergenic regions typically showed short stretches between 8 and 10 bases of near-perfect conservation, surrounded by non-conserved bases, rich in isolated gaps. Protein-coding genes on the other hand were much more uniform in their conservation, and typically differed in the largely-degenerate third-codon position. The proportion of sites corresponding to a different nucleotide in at least one of the three species is 58% in intergenic regions but only 30% in genic regions – a



**Figure 2.2. Patterns of change in genes and intergenic regions.** Schematic representation of two multiple sequence alignments in ORF YMR017W and neighboring intergenic region. Aligned nucleotides across the four species are shown as stacked squares, colored by their conservation: green for conserved positions, yellow otherwise. Alignment gaps are shown in white and frame-shifting insertions (length not a multiple of 3) are shown in red. In addition to the abundance of frame-shift indels shown here, numerous in-frame stop codons are observed in the other three species.

difference of  $\sim 2$ -fold. The difference becomes much greater when one considers the gapped positions in alignments, representing insertion and deletion events (indels). The proportion of indels is 14% in intergenic regions, but only 1.3% in genic regions. The contrast is even sharper for indels whose length is not a multiple of three. These would disrupt the reading frame of a functional protein-coding gene, and are detrimental when they occur in real genes, unless they are compensated by a nearby indel that restores the reading frame. Frame-shifting gaps are found in 10.2% of aligned positions in intergenic regions, but only in 0.14% of positions in genic regions, a 75-fold strong separation. We used these alignment properties to recognize real genes.

### 2.3. Reading Frame Conservation Test

We developed a Reading Frame Conservation (RFC) test to classify each ORF in *S. cerevisiae* as biologically meaningful or not, based on the proportion of the ORF over which reading frame is locally conserved in each of the other three species. Each species with an orthologous alignment cast a vote for accepting or rejecting the ORF, and the votes were tallied to reach a decision for that ORF.

We evaluated the percent of nucleotides that are in the same frame within overlapping windows of the alignment. For every such window, we labeled each nucleotide of the first sequence by its position within a codon, as 1, 2 or 3 in order, starting at codon offset 1. We similarly labeled the nucleotides of the second sequence, but once for every start offset (1, 2, or 3). We then counted the percentage of gapless positions in the alignment that contained the same label in both aligned species, and selected the maximum percentage found in each of the three offsets of the second sequence (Figure 2.3). The final RFC value for the ORF was calculated by averaging the percentages obtained at overlapping windows of 100 nucleotides starting every 50

Scer	CTCTAGATTTCATCTT-GTCGATGTTCAAACAACGTGTTA-----TCAGAGAAACAGCTCTATGAGAAATCAGCTGATG									
Scer_f1	1231231231	2312312312	-12312312312312312312312312	-----	31231231231231231231	23123123123123123123				
Spar	TATTCATA-TCTCATCTTCATCAATGTTCAAACAGCGTGTTACAGACACAGAGAAACAGCTTC-TGAGAAGTCAGCCGGTG									
Spar_f1	12312312	-3123123123	12312312312312312312312312	31231	2312312312312312312	-31231231231231231				→ 43%
Spar_f2	23123123-	1231231231	2312312312312312312312312312312	3123123123123123123		12312312312312312				→ 34%
Spar_f3	31231231-	231231231231231231231231231231231231231231231231	-23123123123123123123							→ 23%

**Figure 2.3. Reading Frame Conservation Test.** Gaps in this alignment between *S. cerevisiae* and *S. paradoxus* change the correspondence of reading frame.

nucleotides. For overlapping ORFs in the *S. cerevisiae* genome ( $n = 948$ ), the RFC was calculated only for the portion unique to each overlapping ORF. For spliced genes ( $n = 240$ ), the RFC was calculated only on the largest exon.

We found that the distribution of frame conservation within each species is bimodal, and we chose a simple cutoff for each species, 80% for *S.paradoxus*, 75% for *S.mikatae* and 70% for *S.bayanus*. If the RFC of the best hit was above the cutoff, a species voted for keeping the ORF tested. If the RFC was below the cutoff and the hit was trusted as orthologous, the species voted for rejecting the tested ORF. Finally, if no orthologous hit could be found due to coverage, a species abstained from voting. We calculated a score between  $-3$  and  $+3$  for every ORF based on the number of species that accepted it ( $+1$ ) and the number of species that rejected it ( $-1$ ). We kept all ORFs with a score of 1 or greater, and rejected all ORFs with a score of  $-1$  or smaller. We manually inspected the remaining ORFs.

We also applied this test to 3966 annotated ORFs with associated gene names (Table 2.4). These have been studied and named in at least one peer-reviewed publication, and are likely to be represent real genes. Only 15 of these (0.38%) were rejected (*KRE20*, *KRE21*, *KRE23*, *KRE24*, *VPS61*, *VPS65*, *VPS69*, *BUD19*, *FYV1*, *FYV2*, *FYV12*, *API2*, *AUA1*, *ICS3*, *UTR5*, *YIM2*). We inspected these manually and concluded that all were indeed likely to be spurious. Most lack experimental evidence. For the remainder, reported phenotypes associated with deletion of the ORF seems likely to be explained by fact that the ORF overlaps the promoters of other known genes.

	Accept	Reject
~4000 named genes	99.6%	0.1%
~300 intergenic regions	1%	99%
2000 Hypothetical genes	1500	500

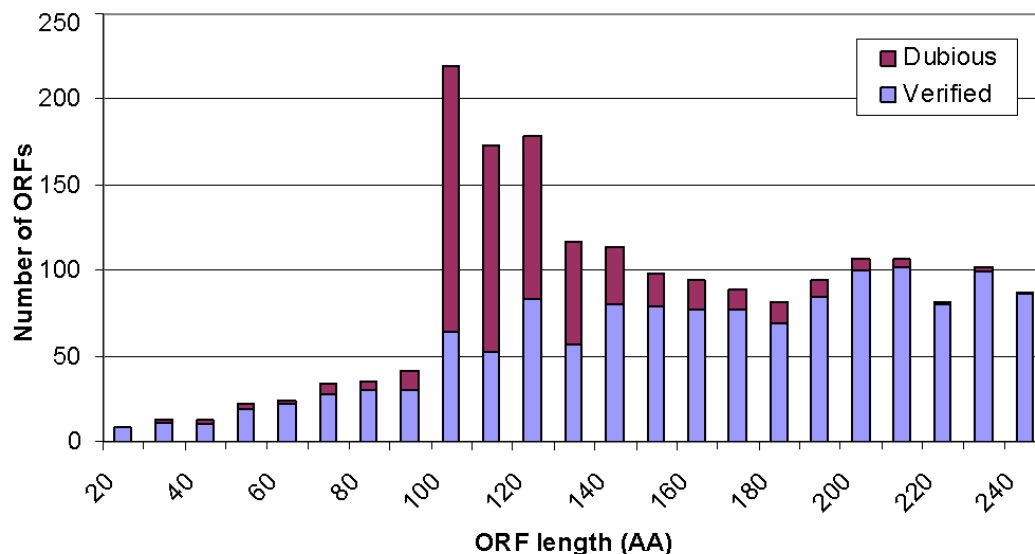
**Table 2.4. Testing all annotated protein-coding genes.** The RFC test showed strong sensitivity and specificity, accepting virtually all experimentally verified genes (named genes) and rejecting all intergenic regions tested. We further applied this test to all the hypothetical genes and showed that more than 500 currently annotated genes are not real.

To investigate the power of the approach to reject spurious ORFs, we also applied it to a set of controls sequences consisting of 340 intergenic sequences in *S. cerevisiae* with lengths similar to the ORFs tested (Table 2.4). About 96% were rejected as having conservation properties incompatible with a biologically meaningful ORF, showing that the test has high sensitivity. Of the remaining 4% that were not rejected, close inspection shows that three-quarters appear to contain true ORFs. Some define short ORFs with conserved start and stop codons in all four species and others extend *S. cerevisiae* ORFs in the 5'- or 3'-direction in each of the other three species. Thus, at most 1% of true intergenic regions failed to be rejected by the RFC test.

The conservation-based gene identification algorithm we proposed has thus high sensitivity and specificity. In the next section, we apply it systematically for de-novo gene identification in *S. cerevisiae*.

#### 2.4. Results: Hundreds of previously annotated genes are not real

When the yeast genome sequence was completed<sup>23</sup>, 6275 ORFs were identified in the nuclear genome that could theoretically encode proteins encoding at least 100 amino acids and that do not overlap a longer ORF by more than half of their length (Figure 2.5). SGD has since updated the catalog based on complete resequencing and re-annotation of chromosome III, re-analysis of other chromosomes and reports in the scientific literature.



**Figure 2.5. Rejected genes are mainly short.** These are likely to be occurring by chance alone given the nucleotide composition of the yeast genome. The rejected genes show no evidence of function, such as mRNA expression, protein function, genetically or bio-chemically.

This resulted in a current version (as of May 2002) with 6062 ORFs  $\geq 100$  amino acids, consisting of 3966 ‘named’ genes (described in at least one publication) and 2096 ‘uncharacterized’ ORFs. SGD also includes a small collection of ORFs  $< 100$  amino acids (see below).

We sought to apply the RFC test to all 6062 ORFs in SGD. A total of 117 could not be analyzed because they were almost completely contained within an overlapping ORF (99 cases, with average non-overlapping portion = 12 bp) or because an orthologous region could not be unambiguously defined in any of the species (18 cases). Of the 5945 ORFs tested, the analysis strongly validated 5550 ORFs. The vote was unanimous in 5458 (~98%) of cases. In the remaining cases, a valid gene appears to have degenerated in one of the four species. A total of 367 ORFs were strongly rejected. These rejections were unanimous in 63% of cases. In most of the remaining cases, *S. paradoxus* was too closely related to *S. cerevisiae* to have accumulated enough frameshifts to allow definitive rejection. The analysis deadlocked (one confirmation, one rejection, one abstention) for 28 ORFs (0.5%). We inspected these, together with the 117 cases that could not be analyzed due to overlaps and found convincing evidence (based on conservation of amino acids, start and stop codons, and presence of indels), that 20 are valid protein coding genes and 105 are spurious. We were unable to reach a judgment in the remaining 20 cases. Overall, a total of 5570 ORFs were accepted, 472 ORFs were rejected, and 20 remain ambiguous.

The vast majority of the rejections (96%) involve uncharacterized ORFs (for an example see Figure 2.6). SGD reports no compelling biological evidence (such as



**Figure 2.6. Example of a rejected gene.** DNA sequence that was previously thought to encode a gene shows an accumulation of frame-shifting insertions and deletions (for color key see Figure 2.2). The sequence in fact does not correspond to a gene, get transcribed, or produce a protein product.



changes in mRNA expression) to suggest that these ORFs encode a true gene. Most of these overlap another well-conserved ORF, but show many insertions and deletions in the non-overlapping portion. The remainder tend to be small (median = 111 aa, with 93%  $\leq$  150 aa) and show atypical codon usage<sup>23,39,40</sup>. Figure 2.6 illustrates the case of an ORF of 333 bp that is clearly biologically meaningless. The orthologous sequence in all four species is laden with frameshifts (as well as stop codons). Only one rejected ORF, YBR184W, appears to represent a true gene that fails the RFC test because it is evolving very rapidly (see section 6.6).

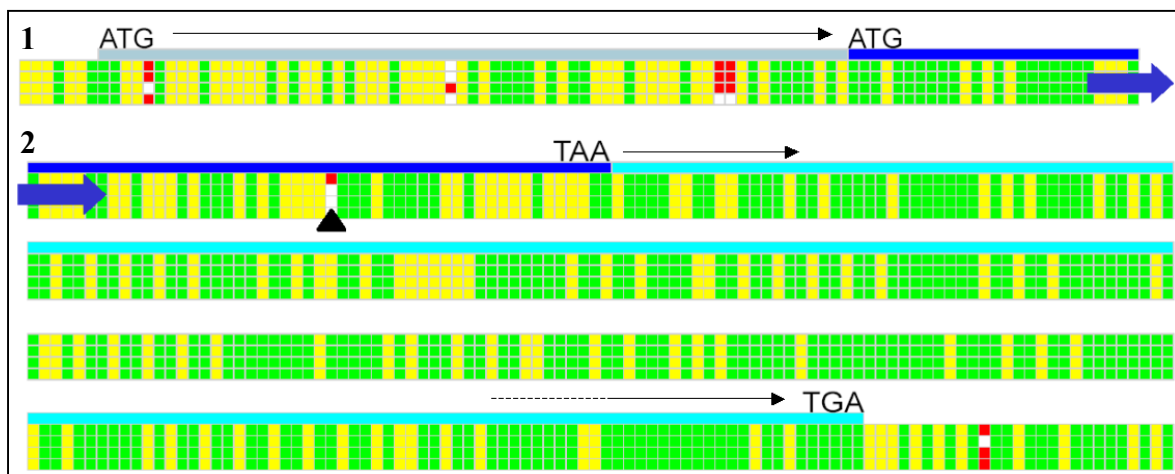
In summary, the Reading Frame Conservation (RFC) test allowed a major revisiting of the yeast genome annotation. By observing the pattern of indels in the multiple alignment of predicted ORFs, it allowed us to automatically classify them as biologically meaningful or spurious. It reached a decision automatically in 98% of cases, accepting 99% of named ORFs and rejecting 99% of real intergenic regions, showing strong sensitivity and specificity. It resulted in a drastic reduction of the yeast gene count, rejecting nearly 500 ORFs. We next use the comparative information to refine the boundaries of ORFs.

## 2.5. Refining Gene Structure

Comparative genome analysis not only improves the recognition of true ORFs, it also yields much more accurate definitions of gene structure – including translation start, translation stop and intron boundaries. We used the comparative data to identify sequencing errors and refine the boundaries of true genes. Previous annotation of *S. cerevisiae* has defined the start of translation as the first in-frame ATG codon. However, the actual start of translation could lie 3' to this point, and the earlier in-frame ATG may be due to chance. Alternatively, if sequencing errors or mutations have obscured an earlier in-frame ATG codon, the true translation start could lie 5' to this point. Similarly, the annotated stop codon could be erroneously annotated, due to sequencing errors. Identifying the correct gene boundaries is important for many reasons, both experimental (for example to construct gene probes), as well as computational (for example to search for regulatory motifs).

We examined the multiple alignment of unambiguous ORFs to identify discrepancies in the predicted start and stop codons across the four species. We searched for the first in-frame ATG in each species and compared it to the annotated ATG in *S. cerevisiae*. In the *S. cerevisiae* start was not conserved, we automatically suggested a changed translation start if a subsequent in-frame ATG was conserved in all species and was the first in-frame ATG in at least one species. Otherwise, we searched for a conserved ATG 5' to that point. Similarly, we suggested changes in stop codons when a common stop in all other species disagreed with the *S. cerevisiae* annotation. We manually inspected the alignments to confirm that the suggested start and stop boundary changes agreed with conservation boundaries. We identified merges of consecutive *S. cerevisiae* ORFs, when they unambiguously matched a single ORF in at least one other species, and when their lengths added up to the length of the matching ORF.

We identified 210 cases in which the presumed translational start in *S. cerevisiae* does not correspond to the first in-frame start codon in at least two of the three other species (Figure 2.7 panel 1). In the vast majority of these cases, inspection of the sequence alignments provides strong evidence for an alternative conserved position for the translational start, either 3' or 5' to the previous annotation. We observed a lower overall conservation as well as frame-shifting indels outside the new boundaries. Similarly, we identified 330 cases in which the presumed translational stop codon in *S. cerevisiae* does not correspond to the first in-frame stop codon in at least two of the three



**Figure 2.7. Refining gene boundaries.** The start and stop codons of more than 300 genes have been refined based on the comparisons. These sometimes reveal sequencing errors in *S. cerevisiae*.

species. In ~25% of these cases, the other three species share a common stop codon and a single base change to the *S. cerevisiae* sequence would result in a stop codon in the corresponding location (Figure 2.7 panel 2). The remaining 75% of cases appear to represent true differences in the location of the translational stop across the species. Thus, stop codons appear to show more evolutionary variability in position than start codons.

We also developed methods for the automatic detection of frame-shifting sequencing errors. When regions of the multiple alignment shifted from one well-conserved reading frame to another well-conserved reading frame, we pinpointed regions of potential sequencing errors in each of the species. A number of these were detected in the reference sequence of *S. cerevisiae*. We confirmed 32 of these computational predictions by resequencing and found that in each case the published sequence was in error, and that the predicted erroneous nucleotide was always within a few base pairs from the experimentally confirmed sequencing error.

We identified 32 cases where two adjacent ORFs in *S. cerevisiae* are joined into a single ORF in all three other species. In every case, a single nucleotide change would suffice to join the ORFs in *S. cerevisiae* (either a substitution altering a stop codon or an indel altering the reading frame). In principle, these cases could represent errors in the genome sequence, mutations private to the sequenced strain S288C, or substitutions fixed in *S. cerevisiae*. We examined 19 cases by resequencing the relevant region in S288C. Our results revealed an error in the published sequence in 11 cases (establishing that there is a single ORF in S288C) and confirmed the published sequence in the remaining 7 cases. Sequencing of additional strains will be required to determine whether these remaining cases represent differences in S288C alone or in *S. cerevisiae* in general.

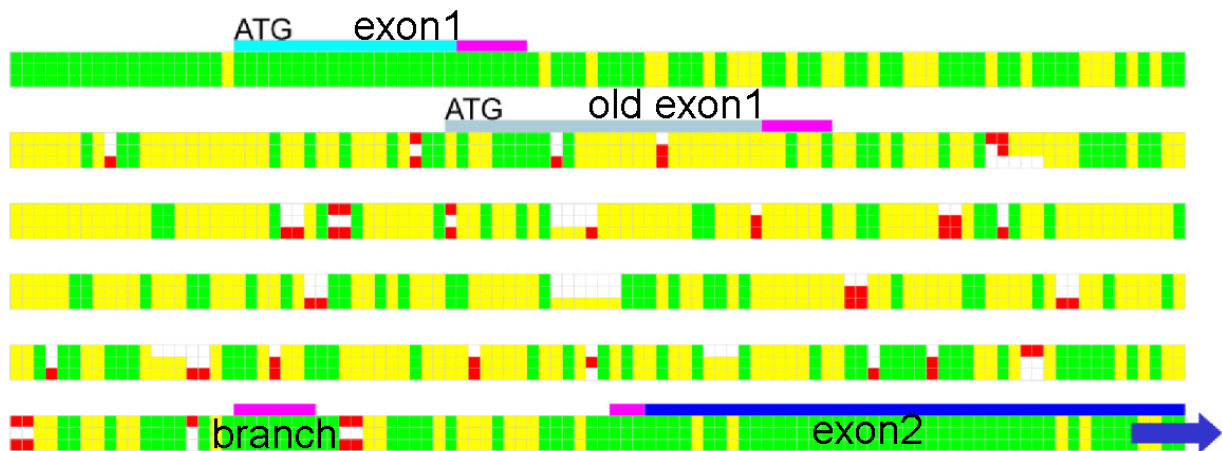
We also found two named ORFs (*FYV5* and *CWH36*) that pass the RFC test and cause phenotypes when deleted, but show no significant protein similarity across the four species. In both cases, inspection reveals that the opposite strand encodes a protein that shows strong amino acid conservation. (The latter gene has two introns, increasing the count of doubly spliced genes to 8.) In each case, we postulate that the protein responsible for the reported deletion phenotype is encoded on the opposite strand.

All merges and boundary refinements suggested specific changes to the nucleotide sequence of *S. cerevisiae* (except 3' changes of translation start that required

no change). To validate our predictions, we re-sequenced the sites of predicted sequence discrepancies. We used both forward and reverse reads in two different PCR reactions spanning the site. We examined 4 cases in which the comparative data suggested an earlier start codon and found, by resequencing, that all correspond to errors in the published sequence of S288C. We examined 17 such cases and found that 15 are explained by errors in the published sequence of S288C.

**New Introns.** We also examined the conservation of introns in the yeast genome. We studied 218 of the 240 ORFs reported in SGD to contain at least one intron (omitting the rest primarily due to lack of an orthologous alignment). In 92% of cases, the donor, branchpoint, and acceptor sites were all strongly conserved with respect to both location and sequence. Moreover, exon boundaries closely demarcated the domains of sequence conservation as measured by both nucleotide identity and absence of indels. Discrepancies were found in 17 cases, of which at least 9 strongly suggest that the previous annotation is incorrect. Five identify a new first exon (Figure 2.8) and four predict that a previously annotated intron is spurious.

We then sought to identify previously unrecognized introns by searching the *S. cerevisiae* genome for conserved splicing signals. We searched for conserved and proximal splice donor and branch signals and manually inspected the resulting alignments. Having constructed multiple alignments of ORFs and flanking intergenic regions, we searched for conserved splicing signals. We used 10 variants of splice donor signals (6-7bp) and 8 variants of branch site signals (7bp) that are found in



**Figure 2.8. Identifying correct splicing.** The short first exon was incorrectly annotated in *S. cerevisiae*. A shorter and earlier first exon is conserved across the four species, and corresponds to the correct splicing.

experimentally validated *S. cerevisiae* introns<sup>41</sup>. We searched each species independently but required that orthologous signals appear within 10 bp from each other in the multiple alignment of the region. We also required that branch and donor be no more than 600bp apart, which is the case for 90% of known *S. cerevisiae* introns. We then inspected the multiple alignment surrounding the conserved signals for three properties: (1) a conserved acceptor signal, [CT]AG, 3' of the branch site (2) high RFC 5' of the donor signal and 3' of the acceptor signal. (3) low RFC within the intron. Roughly half of the conserved donor/branch pairs met our additional requirements.

We predict 58 novel introns. Fifty cases affect the structure of known genes (defining new 5'-exons in 42 cases, 3'-exons in 7 cases and an internal splice in one case) and two indicate the presence of new genes. The relationship of the apparent splice signals to existing genes is unclear for the remaining six cases. We visually inspected our predictions and compared our results to experimental studies by Ares and colleagues that identified new introns using techniques such as microarray hybridization<sup>41</sup>. Of our 58 predicted introns, 20 were independently discovered by this group. Of the four annotated introns predicted to be spurious, all four show no experimental evidence of splicing. Our remaining predictions are currently being tested in collaboration with Ares and colleagues.

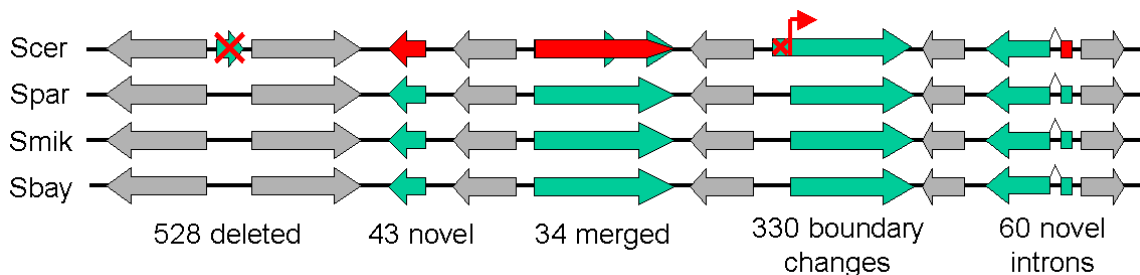
## **2.6. Analysis of small ORFs**

The power of our method was limited for small ORFs. Smaller regions may indeed show lack of indels due to chance, and hence a high reading frame conservation score may not be meaningful.

We tested 141 ORFs encoding 50-99 amino acids for which some biological evidence has been published and are reported in SGD. Applying the RFC test and inspecting the results, we conclude that 120 appear to be true genes, 18 appear to be spurious ORFs and 3 remain unresolved. SGD also lists 32 ORFs encoding < 50 aa. We did not undertake a systematic search for all such ORFs, because control experiments showed that the RFC test lacked sufficient power to prove the validity of such small ORFs (see below). However, it is able to reject 7 of the 32 ORFs as likely to be spurious. Our yeast gene catalogue thus contains 188 short genes (<100 aa), of which 43 are novel.

To evaluate the predictive power of the RFC test for small ORFs, we additionally tested for presence of in-frame stop codons in the other species. When a small ORF in *S. cerevisiae* showed a strong overall frame conservation, we measured the length of the longest ORF in the same orientation in each orthologous locus. We measured the percent of the *S. cerevisiae* length that was open in each species (no stop codons), and took the minimum of the three percentages (OPEN) across the three additional species. When the reading frame was open in each of the other species, the lengths found were identical to that of *S. cerevisiae*, and OPEN was 100%. When OPEN was below 80%, we concluded that stop codons appeared in the orthologous sequence, and therefore that the RFC test falsely accepted a segment that did not correspond to a true gene. We observed the distribution of OPEN for different values of RFC. For *S. cerevisiae* ORFs between 50 and 100 amino acids (aa), selecting for high RFC automatically selected for high OPEN, and we estimated the test has high specificity. For ORFs between 30 and 50 aa however, only a small portion of the ORFs with high RFC show a high OPEN, and we conclude that the lack of indels within the small interval considered is not due to selective pressure, but instead lack of evolutionary distance between the species aligned.

We further systematically searched the remainder of the *S. cerevisiae* genome and evaluated all ORFs in this size range. Control experiments demonstrated that the RFC test has high power to discriminate reliably between valid and spurious ORFs in this size range. The genome contains 3161 such ORFs, nearly all are readily rejected by the RFC test. However, 43 novel genes were identified. These ORFs not only pass the RFC test, but they also have orthologous start and stop codons. Five of these have been reported in the literature subsequent to the SGD release studied here



**Figure 2.9. Revised yeast catalogue.** Our analysis has affected nearly 15% of all genes.

## 2.7. Conclusion: Revised yeast gene catalog

Based on the analysis above, we propose a revised yeast gene catalog consisting of 5538 ORFs  $\geq 100$  amino acids. This reflects the proposed elimination of 503 ORFs (366 from the RFC test, 105 by manual inspection and 32 through merger). A total of 20 ORFs in SGD remain unresolved. Complete information about the gene catalog is provided in <sup>29</sup> and will be discussed more fully in a subsequent manuscript in collaboration with SGD and other yeast investigators. The revised gene count is consistent with at least two recent predictions based on light shotgun coverage of related species<sup>4,5</sup>. We believe that this represents a reasonably accurate description of the yeast gene set, because the analysis examines all ORFs  $\geq 100$  amino acids, the methodology has high sensitivity and specificity and the evidence is unambiguous for the vast majority of ORFs. Nonetheless, some errors are likely to remain. The results could be confirmed and remaining uncertainties resolved by sequencing of additional related yeast species, as well as by other experimental methods.

Despite the intensive study of *S. cerevisiae* to date, comparative genome analysis points to the need for a major revision of the yeast gene catalog affecting more than 15% of all ORFs (Figure 2.9). The results suggest that comparative analysis of a modest collection of species can permit accurate definition of genes and their structure. Comparative analysis can complement the primary sequence of a species and provide general rules for gene discovery that do not rely solely on known splicing signals for gene discovery. Previous studies have shown that such methods are also applicable to the understanding of mammalian genes<sup>42</sup>. The ability to observe the evolutionary pressures that nucleotide sequences are subjected to radically changes our power for signal discovery.

## CHAPTER 3: REGULATORY MOTIF DISCOVERY

### 3.1. Introduction

Regulatory motifs are short nucleotide sequences typically upstream of genes that are used to control the expression of genes, dictating under which conditions a gene will be turned on or off. Direct identification of regulatory elements is more challenging than that of genes. Such elements are typically short (6-15 bp), tolerate some degree of sequence variation and follow few known rules. To date, the majority have been found by experimentation, such as systematic mutation of individual promoter regions; the process is laborious and unsuited for genome-scale analysis.

Computational analysis of single genomes has been successfully used to identify regulatory elements associated with known sets of related genes<sup>7-9</sup>. These methods typically search for frequently-occurring sequence patterns at various distances upstream of coordinately expressed genes, and will be further described in chapter 4. They are however limited by the experimental information available, and hence do not permit a comprehensive direct identification of regulatory elements<sup>43</sup>.

Comparative genomics offers various approaches for finding regulatory elements. The simplest approach is to perform cross-species sequence alignment to find *phylogenetic footprints*, regions of unusually high conservation. This approach has long been used to study promoters of specific genes in many organisms<sup>10,12,44-46</sup> and recently was applied across the entire human and mouse genomes<sup>19</sup>. The genome alignments of the four *Saccharomyces* species can similarly be used to study each yeast gene, to help define promoters and other islands of intergenic conservation (Figure 3.2).

Our interest was to go beyond inspection of individual islands of conservation to construct a comprehensive dictionary of regulatory elements used throughout the genome. We investigated the conservation properties of known regulatory motifs and used the insights gained to design an approach for *de novo* discovery of regulatory motifs directly from the genome.

In this chapter, we develop and apply methods for genome-wide motif discovery. We compare our results to a database of experimentally validated regulatory motifs and rediscover virtually all previously known motifs. In chapter 4 we develop methods for



inferring a candidate function for the motifs discovered making use of biological knowledge about genes, and in chapter 5 we explore their combinatorial interactions.

### 3.2. Regulatory motifs

The current knowledge of gene regulation is based on focused experimental studies of specific examples. The deletion of a transcription factor was shown to disrupt the use of its target genes. Regulatory elements were identified in genetic screens through function-disrupting mutations that reside outside of a protein-coding ORF. Systematic mutagenesis of a particular promoter region (also known as promoter bashing) and testing the resulting effect on gene expression has been used to identify functional blocks in upstream regions of genes. To identify regulatory motifs at a nucleotide level, footprinting methods can be used. These methods expose the bound region to DNA damaging agents that degrade unbound nucleotides, leaving a ‘footprint’ of the transcription factor on the bound and thus protected nucleotides. Finally, even higher resolution information is obtained through crystal structures of transcription factors bound to DNA. These different methods have produced lists of bound sites for each of a small number of well-studied transcription factors.

The sites bound by these factors exhibit sequence similarities that reveal the binding specificity of each factor, and can be represented in a *regulatory motif*.

Representations for these motifs range from consensus sequences listing the nucleotides involved in binding, to weight matrices and graphical models. *Consensus sequences* or *sequence profiles* are the simplest such representation, giving a list of possible bases for each position in the bound site. Some positions are strict and require the presence of a particular nucleotide, others allow for degeneracies. These can be represented compactly using the IUB standard one-letter code (Table 3.1). More complex representations can be used allowing for more detail in the binding specificity.

IUB	Nucleotides	Name	$[p_A, p_C, p_G, p_T]$
A	A	Adenine	[1, 0, 0, 0]
C	C	Cytosine	[0, 1, 0, 0]
G	G	Glutamine	[0, 0, 1, 0]
T	T	Tyrosine	[0, 0, 0, 1]
S	C or G	Strong	$[0, \frac{1}{2}, \frac{1}{2}, 0]$
W	A or T	Weak	$[\frac{1}{2}, 0, 0, \frac{1}{2}]$
R	A or G	PuRine	$[\frac{1}{2}, 0, \frac{1}{2}, 0]$
Y	C or T	pYrimidine	$[0, \frac{1}{2}, 0, \frac{1}{2}]$
M	A or C	aMino group	$[\frac{1}{2}, \frac{1}{2}, 0, 0]$
K	G or T	Keto group	$[0, 0, \frac{1}{2}, \frac{1}{2}]$
B	C or G or T	Not A	$[0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$
D	A or G or T	Not C	$[\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}]$
H	A or C or T	Not G	$[\frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3}]$
V	A or C or G	Not T	$[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0]$
N	A, C, G or T	aNy base	$[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$

**Table 3.1. Degenerate nucleotide code.**

A *weight matrix* representation of a motif of length  $L$  assigns weight vector  $w_i = [w_A, w_C, w_G, w_T]$  to every position  $i$  between 1 and  $L$ . The binding strength of a sequence can be scored against a weight matrix by simply adding up the corresponding scores for each position. In a probabilistic framework, the weights can represent the relative frequencies of each nucleotide in real motifs, multiplying across the corresponding weights gives the probability that a sequence  $s$  matches the motif represented by  $m$ . Alternatively, if log probabilities are used instead, summing across the matrix gives the corresponding log probability. This probability can be compared to the probability of obtaining  $s$  by chance, to obtain a log-likelihood ratio that the sequence matches the motif. Both consensus sequences and weight matrices model the binding contributions of nucleotide position as independent. More complex Bayesian representations for motifs can be used to capture pairwise and multiple dependencies between positions. As the models become more complex however, the increased power comes at a cost, increasing the number of parameters and possibly overfitting data.

Transcription factors have evolved different ways to contact the DNA double helix, and these are reflected in different types of regulatory motifs. Some factors make one long contact with the DNA helix recognizing between 6 and 8 positions, some of which can be degenerate. One such example is the Mbp1 transcription factor involved in the timing of events such as DNA replication during cell division and recognizes the motif ACGCGT. Other factors contact the DNA at two different points, resulting in motifs with two cores, separated by a stretch of unspecified bases. For example, the binding site recognized by Abf1, a general transcription factor involved in silencing and replication, recognizes the motif RTCRYNNNNNACGR. The DNA-binding domains of other factors are made of two identical parts (and hence called *homodimers*), contacting each other and each contacting the DNA helix. The two parts recognize identical sequences, but on opposite strands, and hence result in motifs that are *reverse palindromes* of themselves. One such example is the Gal4 factor involved in galactose metabolism, recognizing CGGNNNNNNNNNNCCG, namely CGG on one strand spaced by 11 nucleotides (one full turn of the double helix) from its reverse complement, CCG.

### 3.3. Extracting signal from noise

Computationally, discovering regulatory motifs amounts to extracting signal from noise. When the motifs searched are expected to be more frequent than other patterns of the same length, one can apply discovery algorithms such as Expectation Maximization (EM) or Gibbs sampling (and others reviewed in ref <sup>9</sup>). These were pioneered by Lawrence and coworkers<sup>47</sup>, and made popular in software programs like MEME<sup>7,48</sup>, AlignACE<sup>8,49,50</sup> or BioProspector<sup>51</sup>. More recent work has extended these methods to incorporate phylogenetic footprinting<sup>45,52-54</sup>. These methods separate the motif discovery problem in two sub-problems. (1) Given a set of starting coordinates  $i_1, \dots, i_n$  in each of the sequences, construct the optimal matrix representation for a motif that starts at each of these positions. (2) Given a matrix representation for a motif  $m$ , find the starting positions of the best matches for that motif in each of the sequences. These algorithms start with a random assignment for the start positions and infer the best matrix, then iterates to improve the assignment of start positions to better match the motif. EM algorithms choose the optimal assignment for each of these rounds of iteration. Gibbs sampling algorithms instead sample amidst the best start positions. Both algorithms converge as long as the motif searched is actually frequent in the sequences searched, since probabilistically, the algorithms will be likely to sample these motifs in their iterative steps, and upon sampling them will converge to include them.

These methods have typically been applied to the upstream sequences of small sets of genes, but are not applicable to a genome-wide discovery. Instead, k-mer counting methods have been used to find short sequences that occur more frequently in intergenic regions, as compared to coding regions in a genome-wide fashion<sup>43</sup>. However, these typically find very degenerate sequences (such as poly-A or poly-T) and have shown limited power to separate regulatory motifs from the mostly non-functional intergenic regions. This is largely due to the small number of functional instances of regulatory motifs, as compared to the large number of non-functional nucleotides. The discovery of regulatory motifs still relies heavily on extensive experimentation.

Comparative genomics provides a powerful way to distinguish regulatory motifs from non-functional patterns based on their conservation. In this chapter we first study conservation properties of known regulatory motifs. We use these to construct three tests

to detect the genome-wide signature of motif-like conservation. We use these tests to detect all significant patterns with strong genome-wide conservation, constructing a list of 72 genome-wide motifs. We compare this list against previously identified regulatory motifs and show that our method has high sensitivity and specificity, detecting most previously known regulatory motifs, but also a similar number of novel motifs. In chapter 4, we assign candidate functions to these novel motifs, and in chapter 5, we study their combinatorial interactions.

### 3.4. Conservation properties of known regulatory motifs

We first studied the binding site for one of the best studied transcription factors, Gal4, whose sequence motif is CCG(N)<sub>11</sub>CCG (which contains 11 unspecified bases). Gal4 regulates genes involved in galactose utilization, including the *GAL1* and *GAL10* genes that are divergently transcribed from a common intergenic region (Figure 3.2). The Gal4

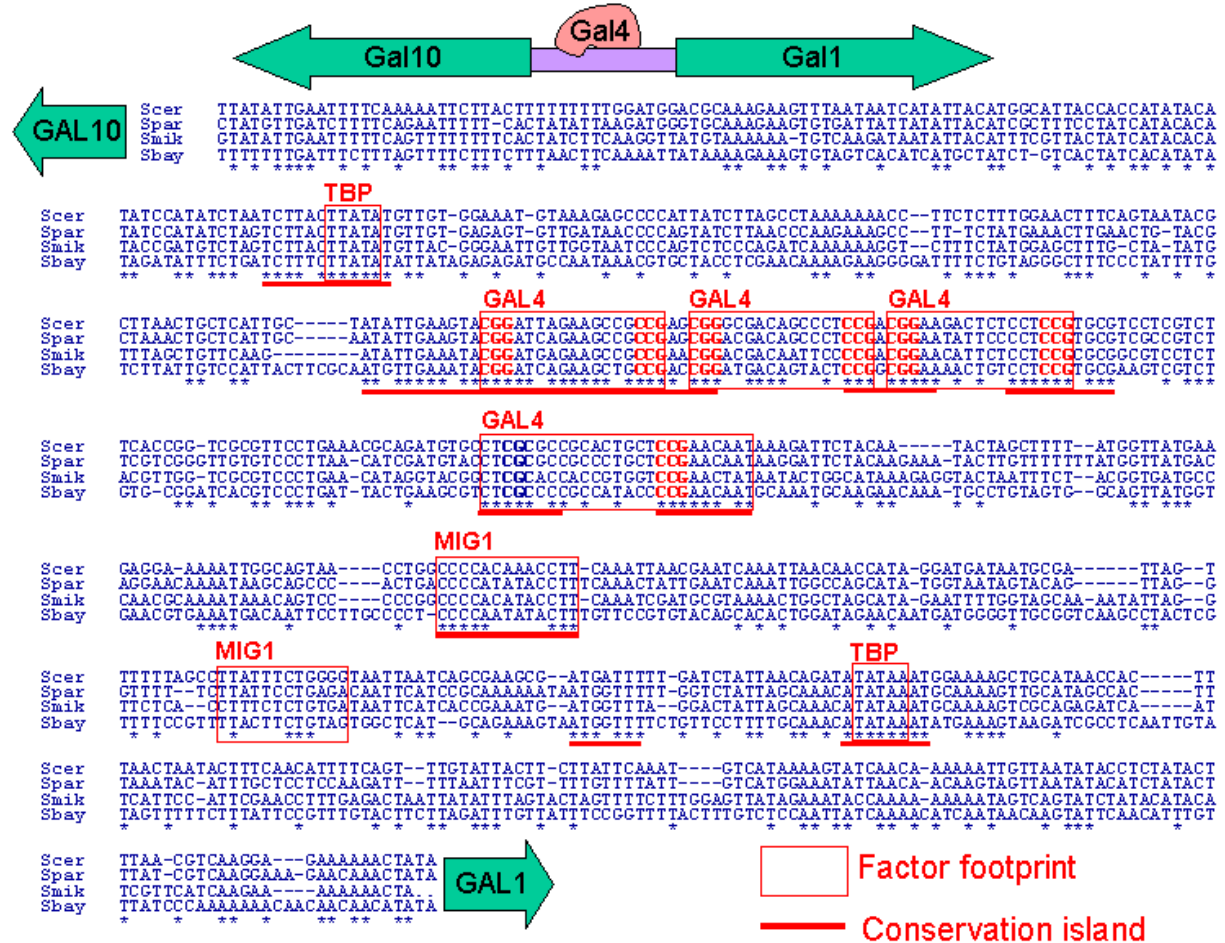
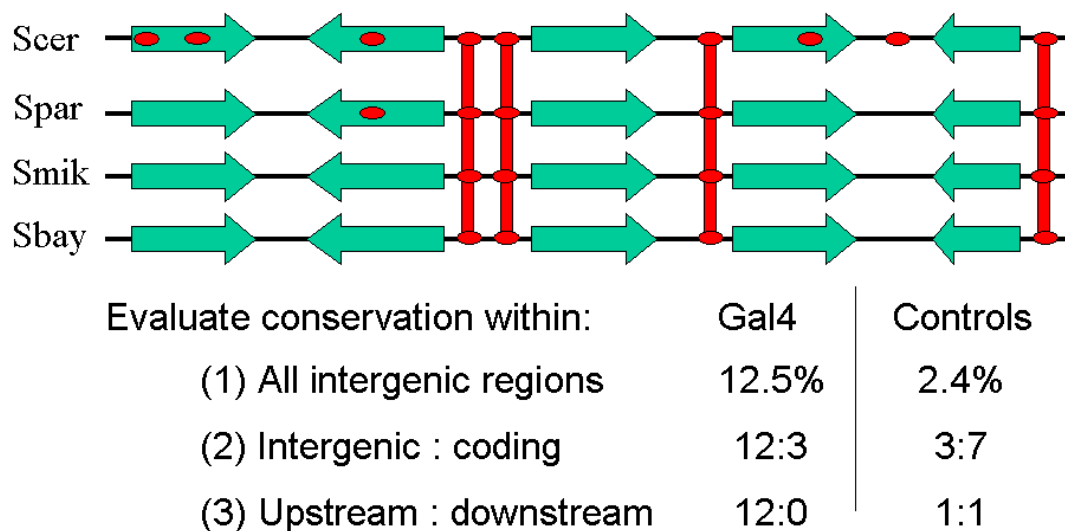


Figure 3.2. Phylogenetic footprinting of the Gal1-Gal10 intergenic region reveals functional nucleotides.

motif occurs three times in this intergenic region, and all three instances show perfect conservation across the four species. In addition, there is a fourth, experimentally validated binding site<sup>55</sup> for Gal4 that differs from the consensus by one nucleotide in *S. cerevisiae*. This variant site is also perfectly preserved across the species.

We then examined the frequency and conservation of Gal4 binding sites across the aligned genomes (Figure 3.3). In *S. cerevisiae*, the Gal4 motif occurs 96 times in intergenic regions and 415 times in genic (protein coding) regions. The motif displays certain striking conservation properties. First, occurrences of the Gal4 motif in intergenic regions have a conservation rate (proportion conserved across all four species) that is ~5-fold higher than for equivalent random motifs (12.5% vs. 2.4%). Second, intergenic occurrences of the Gal4 motif are more frequently conserved than genic occurrences (12.5% vs. 3%). By contrast, random motifs are less frequently conserved in intergenic regions than genic regions (3.1% vs. 7.0%), reflecting the lower overall level of conservation in intergenic regions. Thus, the relative conservation rate in intergenic vs. genic regions is ~11-fold higher for Gal4 than for random motifs. Third, the Gal4 motif shows a higher conservation rate in divergent vs. convergent intergenic regions (those that lie upstream vs. downstream of both flanking genes); no such preferences is seen for control motifs. These three observations suggest various ways to discover motifs based on their conservation properties (see conservation criteria below).



**Figure 3.3. Genome-wide conservation of the Gal4 motif.** The six-fold to 11-fold separation between the conservation of Gal4 and that of random control motifs suggests three signatures for motif discovery.

We extended these observations by assembling a catalog of 55 known regulatory sequence motifs (Table 3.4), by starting with two public databases (SCPD<sup>56,57</sup> and YTFD<sup>58</sup>) and curating the entries to select those with the best support in the literature.

Factor	Known motif		Discovered motif		
	Motif	MCS	Motif	Genome-wide	Category- based MCS
ABF1	<b>RTCRYnnnnnACG</b>	50.0	<b>RTCRYknnnnACGR</b>	S	S 36.2
UME6	<b>TCGGCGGCTA</b>	20.9	<b>TSGGCGGCTA</b> WW	S	NC 23.4
CBF1	<b>RTCACRTG</b>	19.0	<b>RTCACGTG</b> V	S	S 17.6
NDT80	<b>TCGGCGGCTDW</b>	18.6	<b>TSGGCGGCTA</b> WW	S	NC 23.4
REB1	<b>TTACCCGG</b>	17.8	<b>RTTACCCGRM</b>	S	S 34.3
MCM1a	<b>TTWCCnWWWRGGAAA</b>	16.5	<b>TTCCnaAttnGGAAA</b>	S	S 13.8
SWI6	<b>ACGCGT</b>	16.4	<b>WCGCGTCGCGt</b>	S	S 10.2
PHO4	<b>CACGTG</b>	16.1	<b>RTCACGTG</b> V	S	S 17.6
MBP1	<b>ACGCGTnA</b>	14.8	<b>WCGCGTCGCGt</b>	S	S 10.2
SWI4	<b>TTTTCGCG</b>	12.4	<b>WTTTCGCGTT</b>	S	S 12.0
DAL81	GATAAG	12.1	—	—	NE —
RPN4	<b>TTTTGCCACC</b>	11.5	<b>TTTTGCCACCG</b>	S	NC 11.0
MSN2	<b>CCCCT</b>	11.3	h <b>RCCCCYTWDt</b>	S	NE 7.8
MSN4	<b>CCCCT</b>	11.3	h <b>RCCCCYTWDt</b>	S	NE 7.8
PDR1	<b>CCGCGG</b>	9.3	<b>YCCGSGGS</b>	S	NE 6.7
ESR2	<b>AAAAWTTTT</b>	8.9	<b>GRRAAAWTTTTCACT</b>	S	NC 15.6
MIG1	<b>CCCCRSWWWW</b>	8.7	<b>DCCCCGCGH</b>	S	NE 8.2
MIG1b	<b>CCCCGC</b>	8.4	<b>DCCCCGCGH</b>	S	NE 8.2
BAS1	<b>TGACTC</b>	8.3	<b>ATGACTCWT</b>	S	S 6.1
GCN4	<b>ATGACTCAT</b>	8.2	<b>ATGACTCWT</b>	S	S 6.1
GAL4	<b>CGGnnnnnnnnnnCGG</b>	8.0	<b>CGGcnnMGnnnnnnnnCGC</b>	S	S 5.0
HSF1b	<b>TTCTAGAA</b>	7.8	<b>TTCTMGAAGA</b>	S	S 7.0
ESR1	<b>GATGAG</b>	7.7	gc <b>GATGAG</b> mtgaraw	S	NC 24.7
MET31	<b>AAACTGTGGC</b>	6.8	<b>SKGTGGSGc</b>	S	S 8.1
AFT1	<b>YRCACCCR</b>	6.8	<b>RVACCCTD</b>	S	NC 10.3
TEA1	CGGnCGG	6.8	—	—	NC —
PUT3	<b>CGGnnnnnnnnnnCGG</b>	6.2	<b>CCGMnnnnnnnnnnSGR</b>	W	NE 5.4
HAP2	<b>TGATTGGC</b>	5.7	<b>TGATTGGT</b>	—	S [6.4]
RAP1	<b>ACACCCATACATTT</b>	5.2	<b>ACACCCACACATnnC</b>	S	S 9.9
LEU3	<b>CCGGnnCCGG</b>	4.9	<b>CCSGTAnCGG</b>	S	S 6.5
MCM1b	<b>YTTCTTAATTWGnnCn</b>	4.8	<b>TTCCnaAttnGGAAA</b>	S	S 13.8
INO4	<b>CATGTGAAAT</b>	4.1	Gnnn <b>CATGTGAA</b>	—	S [6.8]
INO2	<b>CATGTGAAAT</b>	4.1	<b>CATGTG</b>	—	S [4.4]
GLN3	GATAAK	3.8	—	—	NE —
ADR1	GGAGA	3.7	—	—	NE —
FKH2	<b>TTGTTTACST</b>	3.6	<b>tTTGTTTACnTTT</b>	S	S 10.8
FKH1	<b>TTGTTTACST</b>	3.6	<b>tTTGTTTACnTTT</b>	S	S 10.8
RLM1	<b>CTAWWWWTAG</b>	3.6	<b>CTAnnTTTAG</b>	S	S [4.7]
SWI5	<b>KGCTGR</b>	3.4	<b>TGCTGG</b>	—	S [6.1]
HAP1	CGGnn <b>TAnCGG</b>	2.5	<b>GCnnTAnCGG</b>	S	NC 4.8
XBP1	<b>MCTCGARRRnR</b>	2.5	<b>TCTCGARRA</b>	S	NC 12.5
MAC1	<b>TTTGCTCA</b>	2.3	<b>TGCTCA</b>	—	S [5.4]
TBF1	<b>TTAGGG</b>	2.3	<b>GKBAGGGT</b>	S	NC 4.8
MSE	<b>TTTTGTG</b>	1.4	<b>TTTTGTGTCRC</b>	S	NC 9.9
STE12	<b>RTGAAACA</b>	0.7	<b>YTGAAACA</b>	—	S [12.2]
DIG1	<b>RTGAAACA</b>	0.7	<b>YTGAAACA</b>	—	S [12.2]
MET4	<b>TGGCAAATG</b>	0.7	<b>CGGTGGCAAAA</b>	S	NE —
HAP4	<b>TnRTTGGT</b>	0.5	<b>TGATTGGT</b>	—	S [6.4]
SMP1	ACTACTA <b>WWWWTAG</b>	0.4	—	—	NE —
ACE2	<b>GCTGGT</b>	-0.6	<b>TGCTGGT</b>	—	S [7.4]
YAP1	TTACTAA	-1.1	—	—	NE —
CIN5	TTACTAA	-1.1	—	—	NE —
RME1	GAACCTCAA	-1.4	—	—	NE —
HAC1	CAGCGTG	-1.4	—	—	NC —
GCR1	<b>GGAAG</b>	-18.5	<b>GGAAGC</b>	—	S [4.4]

**Table 3.4. Genome-wide conservation of known motifs.** Matching nucleotides in bold. S=strong match, W=weak match, NE=not enriched, NC=no category available. Category scores in brackets.

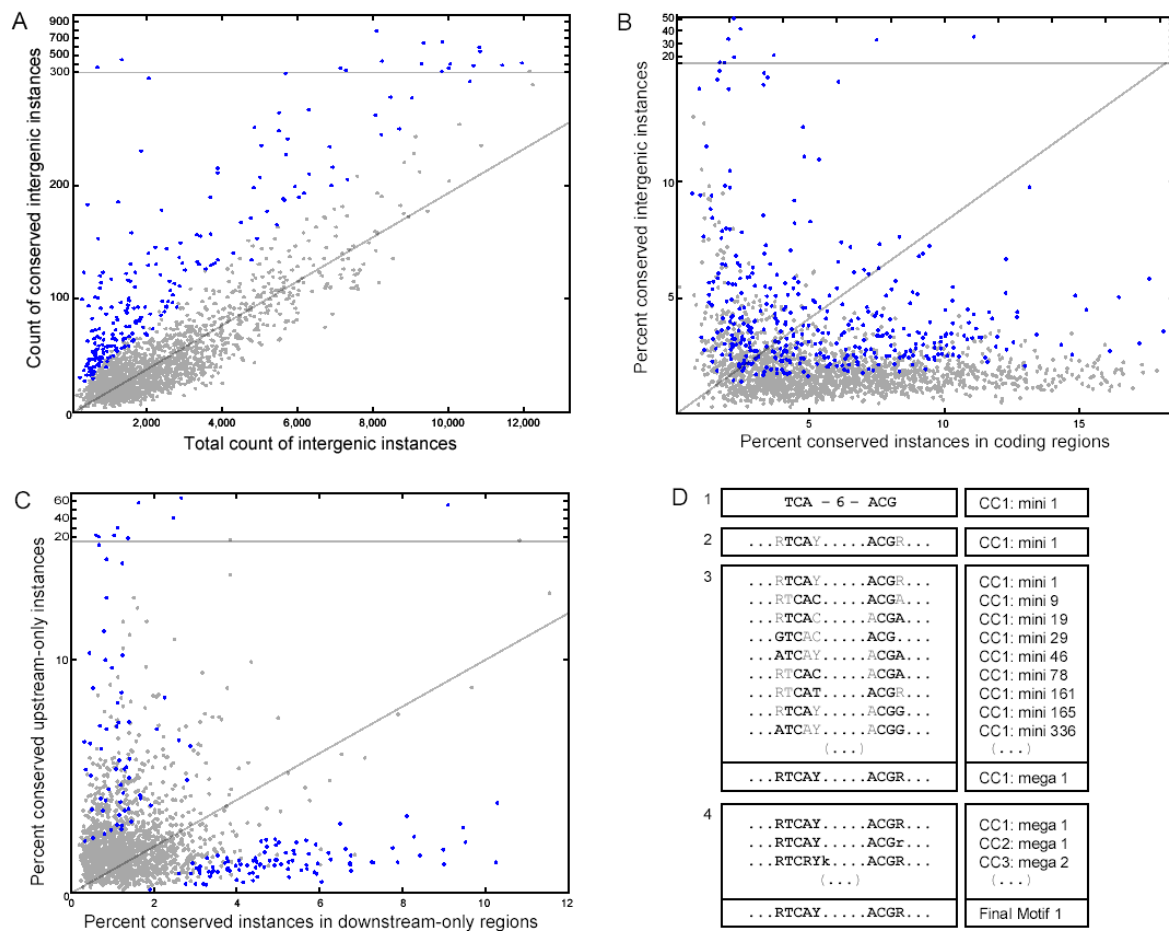
We defined a Motif Conservation Score (MCS) based on the conservation rate of the motif in intergenic regions. To evaluate the Motif Conservation Score (MCS) of a motif  $m$  of given length and degeneracy, we compared its conservation ratio to that of random patterns of the same length and degeneracy. We first computed the table  $F$  containing the relative frequencies of two-fold and three-fold degenerate bases, given the *S. cerevisiae* nucleotide frequencies (.32 for A and T, .18 for C and G). For example,  $W=[AT]$  (.32\*.32) is a more likely two-fold degenerate base than  $Y=[CT]$  (.18\*.32). We then selected 20 random intergenic loci in *S. cerevisiae*. For each of these loci, we used the order of nucleotides at that locus together with the order of degeneracy levels in  $m$  to construct a random motif. If the first character of  $m$  was two-fold degenerate and the first nucleotide at the selected locus was A, we picked a two-fold degenerate base containing A (W, R or M), their relative frequencies dictated by  $F$ , and continued for every character of  $m$ . We then counted conserved and non-conserved instances of each of the 20 generated control patterns and computed  $r$ , the log-average of their conservation rates. We then counted the number of conserved and non-conserved intergenic instances of  $m$ , and computed the binomial probability  $p$  of observing the two counts, given  $r$ . We finally reported the MCS of the motif as a z-score corresponding to  $p$ , the number of standard deviations away from the mean of a normal distribution that corresponds to tail area  $p$ . Nearly all of these sequence motifs are binding sites of known transcription factors. Most of the known motifs show extremely strong conservation, with 60% having  $MCS \geq 4$  (which is substantially higher than expected by chance). Some of the motifs, however, show relatively modest MCS. These motifs may be incorrect, suboptimal or not well conserved.

### 3.5. Genome-wide motif discovery

Our methodology for genome-wide motif discovery involves first identifying conserved *mini-motifs* and then using these to construct full motifs (Figure 3.5). Mini-motifs are sequences of the form  $XYZn_{(0-21)}UVW$ , consisting of two triplets of specified bases interrupted by a fixed number (from 0 to 21) of unspecified bases. Examples are TAGGAT, ATAnnGGC, or the Gal4 motif itself. The total number of distinct mini-motifs is 45,760, if reverse complements are grouped together.

Conserved mini-motifs are evaluated according to three conservation criteria (CC1-3), based on our observations about the properties of the Gal4 motif. In each case, conservation rates are normalized to appropriate random controls. CC1 (Intergenic conservation) evaluates the conservation rate of a mini-motif in intergenic regions. CC2 (Intergenic-genic conservation) evaluates the stronger conservation in intergenic regions as compared to genic regions. CC3 (Upstream-downstream conservation) evaluates the different conservation of a mini-motif when it occurs upstream vs. downstream of a gene.

CC1: Intergenic conservation. We searched for mini-motifs that show a significant conservation in intergenic regions. For every mini-motif, we counted *ic* the number of perfectly conserved intergenic instances in all four species, and *i* the total number of intergenic instances in *S.cerevisiae*. We found that the two counts seem linearly related for the large majority of patterns (Figure 3.5 panel A), which can be



**Figure 3.5. Genome-wide motif discovery method.** The three conservation tests and motif collapsing.



attributed to a basal level of conservation  $r$  given the total evolutionary distance that separates the four species compared. We estimated the ratio  $r$  as the log-average of non-outlier instances of  $ic/i$  within a control set of all motifs at a given gap size. We then calculated for every motif the binomial probability  $p$  of observing  $ic$  successes out of  $i$  trials, given parameter  $r$ . We assigned a z-score  $S$  to every motif corresponding to probability  $p$ . This score is positive if the motif is conserved more frequently than random, and negative if the motif is diverged more frequently than random. We found that the distribution of scores is symmetric around zero for the vast majority of motifs. The right tail of the distribution however is much further than the left tail, containing 1190 motifs more than 5 sigma away from the mean, as compared to 25 motifs for the left tail. By comparing the two counts, we estimated that 94% of these 1190 motifs are non-random in their conservation enrichment.

CC2: Intergenic-genic conservation. We searched for motifs that are preferentially conserved in intergenic regions, as compared to coding regions. In addition to  $ic$  and  $i$  (see previous section), we counted the number of conserved coding instances  $gc$ , and the number of total coding instances  $g$ , for every mini-motif. We observed the ratio of conserved instances that are intergenic  $a=ic/(ic+gc)$ , and compared it to the total ratio of motif instances that are intergenic  $b=i/(i+g)$ . Not surprisingly, we found that typically  $b=25\%$  of all motif instances appeared in intergenic regions, which account for roughly 25% of the yeast genome. Similarly, only  $a=10\%$  of conserved motif instances appeared in intergenic regions, which reflects the lower conservation of intergenic regions. To correct for this typical depletion in intergenic conservation, we estimated a correction factor  $f=a/b$  for mini-motifs of similar GC-content. Then for a given mini-motif, the proportion of all instances found in intergenic regions and the correction for the lower conservation of intergenic regions together gave us  $r=f*i/(i+g)$ , the expected ratio of conserved intergenic instances for that motif. We evaluated the binomial probability  $p$  of observing at least  $ic$  conserved instances in intergenic regions and  $ic+gc$  conserved instances overall, given the expected ratio  $r$ . As in CC1, we computed a z-score  $S$  for every motif and found a distribution centered around zero for the large majority of motifs, and a heavier right tail. We selected 1110 motifs above 5 sigma and estimated that 97% are non-random as compared to only 39 motifs below -5 sigma.

CC3: Upstream-downstream conservation. We searched for motifs that are differentially conserved in upstream regions and downstream regions. We defined upstream-only intergenic regions in divergent promoters that are upstream of both flanking ORFs, and downstream-only intergenic regions in convergent 3' terminators that are downstream of both flanking ORFs. We then counted ***uc*** and ***u***, the conserved and total counts in upstream-only regions, and similarly ***dc*** and ***d*** in downstream-only regions. We found that upstream-only and downstream-only regions have similar conservation rates, and the ratios ***uc/u*** and ***dc/d*** are both similar to ***ic/i*** for the large majority of motifs. We thus used a simple chi-square contingency test on the four counts (*uc,u,dc,d*) to find motifs that are differentially conserved. We found 1089 mini-motifs with a chi-square value of 10.83 or greater, which corresponds to a p-value of .001. Given the multiple testing of 45760 mini-motifs, we estimated that roughly 46 will show such a score by chance and that 96% of the selected motifs will be non-random.

The conserved mini-motifs are then used to construct full motifs (Figure 3.5). They are first extended, by searching for nearby sequence positions showing significant correlation with a mini-motif. The extended motifs are then clustered, merging those with substantially overlapping sequences and those that tend to occur in the same intergenic regions. Finally, a full motif is created by deriving a consensus sequence (which may be degenerate). Motifs are typically degenerate, and a single full-motif can be responsible for multiple strong mini-motifs. We now describe methods to recover the full motifs and their degeneracy.

We extended each mini-motif selected by searching for surrounding bases that are preferentially conserved when the motif is conserved. We used an iterative approach adding at every iteration one base that maximally discriminates the neighborhood of conserved motif instances from the neighborhood of non-conserved motif instances. The added base was selected from fourteen degenerate symbols of the IUB code (A, C, G, T, S, W, R, Y, M, K, B, D, H, V). When no such symbol separated the conserved and non-conserved instances with significance above 3 sigma, we terminated the extension. Figure 3.5 panel D shows the top-scoring mini-motif found in CC1 (Row 1), and the corresponding extension (Row 2). We found that many mini-motifs have the same or similar extensions, and we grouped these based on sequence similarity. We measured the

similarity between two motifs as the number of bits in common in the best ungapped alignment of the two motifs, divided by the minimum number of bits contained in either

No.	Discovered motif	Location	MCS	Best category	CCS	Interpretation
1	YCGTnnnnnRYGAY	5'	36.2	ChIP: Abf1	90	Known: Abf1
2	RTTACCCGRM	5'	34.3	ChIP: Reb1	38	Known: Reb1
3	gcGATGAGmtgaraw	5'	24.7	Exp.: cluster 74	62	Known: Esr1 GATGAG
4	TSGGCGGCTAWWW	5'	23.4	GO: meiosis	10	Known: Ume6/Ndt80
5	RTCACGTGV	5'	17.6	ChIP: Cbf1	27	Known: Cbf1/Pho4
6	WTATWTACADG	3'	17.4	Exp.: cluster 16 downstream	25	New: mitochondrial downstream
7	GRRAAAWTTTCACT	5'	15.6	Exp.: cluster 74	37	Known: Esr2
8	TTCnnaAttnGGAAA	5'	13.8	ChIP: Mcm1	29	Known: Mcm1
9	CGTTTCTTTTCY.	5'	13.5	GO: filamentation	7	New: filamentation
10	TYTTCGAGA.	5'	12.5	Exp.: cluster 86	5	Known: Xbp1 (Hsf1-co-ocuring)
11	TTTTCGCG	5'	12.0	ChIP: Swi4	21	Known: Swi4 fixed gap
11a	TTTT = CGCG	5'	12.0	ChIP: Swi4	-	New: Swi4 variable gap
12	TKACGCGTT	5'	12.0	ChIP: Mbp1	18	Known: Mbp1/Swi6
13	STGCGGnnnttTCTnnG	5'	11.8	GO: filamentation	11	New: filamentation
14	YCTATTGTT	5'	11.5	ChIP: Fkh2	6	New: Rlm1-like
15	TTTTGCCACCG	5'	11.0	GO: proteolysis	25	Known: Rpn4/Met4
16	tTTGTTTACnTTT	5'	10.8	ChIP: Fkh2	28	Known: Fkh1/2
17	RVACCCTD	5'	10.3	-	-	Known: Aft1
18	WCGCGTCGCGt	5'	10.2	ChIP: Mbp1	17	New: double Mbp1
19	GGGTnACCC	5'	10.0	ChIP: Reb1	8	New: Reb1 palindrome
20	GnnATGTGTGGGTGT	5'	9.9	ChIP: Fhl1	5	Known: Rap1
21	TTTTGTGTCRC	5'	9.9	ChIP: Sum1	14	Known: Mse
22	TTTCAnCGCGC	5'	9.8	-	-	New: no category
23	TATTAWTATTATtMthatta	3'	9.5	-	-	New: no category
24	SCGnHGGs	5'	8.8	GO: filamentation	6	New: filamentation
25	ACAGCCGCRY	5'	8.6	Exp.: cluster 37	6	New: expression cluster 37
26	DCGCGGGGH	5'	8.1	Exp.: cluster 46	8	Known: Mig1b
27	SKGTGGSGc	5'	8.1	ChIP: Met31	5	Known: Met31
28	TTTTn(19)GCKCG	5'	7.8	-	-	Known: no category
29	HRCCCYTWDt	5'	7.8	Exp.: cluster 8	22	Known: Msn2/4
30	TKCCnnnnGGG	5'	7.3	ChIP: Mcm1	15	Known: Mcm1 (hits tRNA)
31	GTGTCAAGTAAt	5'	7.1	ChIP: Sum1	15	New: Sum1
32	RGTTTTTCCG	5'	7.1	ChIP: Rgt1	7	New: Rgt1
33	TTCTMGAAGA	5'	7.0	ChIP: Hsf1	10	Known: Hsf1
34	YCCGSGGS	5'	6.7	GO: filamentation	9	New: filamentation
35	CnCCTTTTATAC	5'	6.5	-	-	New: no category
36	CCSGTAnCGG	5'	6.5	ChIP: Leu3	8	Known: Leu3
37	SKTKCCTT	5'	6.4	GO: filamentation	7	New: filamentation
38	CTCCCCCTTAT	5'	6.4	Exp.: cluster 8	11	Known: Msn2/4
39	GCCCCG	5'	6.3	GO: filamentation	10	New: filamentation
40	SGCGCGRB	5'	6.3	-	-	New: no category
41	CTCSGCS	5'	6.2	-	-	New: no category
42	TGnKAGCGCCG	5'	6.2	-	-	-
43	ATGACTCWT	5'	6.1	ChIP: Gcn4	44	Known: Gcn4/Bas1
44	CCGAnnnTCGG	5'	6.1	Exp.: cluster 46	6	New: facilitators palindrome
45	SCGMnnnnnnKCG	5'	6.0	-	-	New: no category
46	CnCCGCGCnnTTTs	5'	6.0	-	-	New: no category
47	TTTTnnnnnnnnnnnnGGGT	5'	5.8	-	-	New: no category
48	TGTRnCAW	3'	5.5	-	-	New: no category
49	YCSknnnnnnnnnKCGG	5'	5.4	Exp: cluster 46	6	Known: Put3
50	CGGnnnnnnnnnnnnKCGV	5'	5.4	-	-	New: no category
51	WGTGACg	5'	5.3	ChIP: Sum1	14	New: Sum1
52	RTCCCTV	5'	5.3	-	-	New: no category
53	YTGGTTTAGG	5'	5.2	GO: lipid metabolism	5	New: lipid metabolism
54	TYCGKRM	5'	5.2	GO: filamentation	7	New: filamentation
55	CGCnnnnnnnnnnnnBCGB	5'	5.1	-	-	New: no category
56	TWCCCCM	5'	5.0	Exp.: cluster 46	7	Known: Mig1 + facilitators
57	CGGCnnMGnnnnnnCGC	5'	5.0	ChIP: Gal4	7	Known: Gal4
58	CCGSnnnnnGVC	5'	5.0	-	-	New: no category
59	TRTAMATAKWT	3'	4.8	ChIP: Dig1	7	New: Ste12 (hits tRNA)
60	TtTATAnTATATAnA	3'	4.8	Exp: cluster 74 downstream	6	New: downstream cluster 74
61	GKBAGGGT	5'	4.8	GO: glycolysis	6	Known: Tbf1/new: glycolysis
62	GCnnTTAnCGG	5'	4.8	-	-	Known: Hap1
63	GGCSnnnnnGnnnCGCG	5'	4.7	ChIP: Mbp1	6	Known: Mbp1-like
64	TTCTCnnnnnnCGC	5'	4.7	GO: filamentation	6	New: filamentation
65	SCGKnnnnKCGD	5'	4.5	-	-	New: no category
66	AATATTCTT	3'	4.4	Exp.: cluster 46 downstream	5	New: downstream facilitators
67	CGCGTnnnnnnnnACG	5'	4.4	ChIP: Swi4	8	New: Swi4-vary gap
68	CCGHVGGM	5'	4.3	-	-	New: no category
69	CGCG = TTTT	5'	4.3	-	-	New: no category
70	CGCGnnnnnGGGS	5'	4.2	Exp.: cluster 46	6	New: expression cluster 46
71	CTGCAGGGR	5'	4.2	GO: filamentation	6	New: filamentation

**Table 3.6. Discovered motifs and associated function.**

motif. Based on the pairwise motif similarity matrix, we clustered the extended motifs hierarchically, collapsing two groups if the average similarity between their member motifs was at least 70%. We then computed a consensus sequence for every cluster of extended motifs, resulting into a smaller number of mega-motifs for each test (332 for CC1, 269 for CC2 and 285 for CC3). Row 3 shows the first 9 members of the top cluster in CC1, and the resulting mega-motif. Finally, we merged mega-motifs based on their co-occurrence in the same intergenic regions (Row 4). We computed a hypergeometric co-occurrence score between the intergenic regions hit by each mega-motif and again collapsed these hierarchically. We computed a consensus for every cluster, and iterated the co-occurrence-based collapsing step (results not shown). We obtained fewer than 200 distinct genome-wide motifs. Each full motif is assessed for genome-wide conservation by calculating its MCS, and those motifs with  $MCS \geq 4$  are retained. Each full motif was also tested for enrichment in upstream vs. downstream regions, by comparing its conservation rate in divergent vs. convergent intergenic regions.

### **3.7. Results and comparison to known motifs**

The vast majority of the 45,760 possible mini-motifs show no distinctive conservation pattern. However, ~2400 mini-motifs show high scores by one or more of these criteria (Figure 3.5 panels A, B, C). There is substantial overlap among the mini-motifs produced by the three criteria, with about 50% of those found by one criterion also found by another.

The conserved mini-motifs give rise to a list of 72 full motifs having  $MCS \geq 4$  (Table 3.6). We omit full motifs with low MCS scores, and those that overlap tRNA genes and may be due to secondary RNA structure. Most of the motifs show preferential enrichment upstream of genes, but six are enriched downstream of genes. These 72 discovered motifs, found with no prior biological knowledge, show strong overlap with 28 of the 33 known motifs having  $MCS \geq 4$ . They include 27 strong matches and 1 weaker match. The 72 discovered motifs also contain matches to 8 of the 22 known motifs with  $MCS < 4$ . In these cases, the comparative analysis identified closely related motifs that have higher conservation scores than the known motifs and occur largely at the same genes; these may represent a better description of the true regulatory element. Comparative genomic analysis thus automatically discovered 36 motifs with matches to

most of the known motifs (65% of the full set, 85% of those with high conservation). It also identified 42 additional ‘novel’ motifs not found in our list of known motifs. In the next chapter, we develop methods to understand these novel motifs and assign a candidate function to each of them.

### **3.8. Conclusion**

Motif discovery amounts to extracting small sequence signals hidden within largely non-functional intergenic sequences. This problem is difficult in a single genome where the signal-to-noise ratio is very small. Previous methods have thus been limited to discovering motifs within small sets of genomic regions. We have conducted a genome-wide exhaustive search for all regulatory motifs. We produced a list of 72 strongly conserved motifs, that includes most previously identified motifs. This ability to directly discover regulatory motifs drastically changes our view of gene regulation. Instead of a case-by-case study, we can now observe complete views of all regulatory building blocks. Our method has re-discovered most previously known regulatory motifs without use of any prior biological function. It should theoretically be applicable to any genome for which no experimental data is available. Additionally, in yeast, we can use the biological information to discover the function of the discovered motifs. We can also use biological function to discover additional motifs. These two goals will be the topic of the next chapter.

## CHAPTER 4: REGULATORY MOTIF FUNCTION

### 4.1. Introduction

In response to environmental changes, a single transcription factor can induce the expression of all genes necessary to fulfill a particular function, such as galactose import and utilization. These genes are typically scattered throughout the genome and targeted by the presence in their upstream regions of a specific regulatory motif recognized by the factor. This regulatory motif will be *enriched* in the upstream regions of these genes, namely it will occur more frequently in these regions than expected by chance as compared to the rest of the genome.

This enrichment of regulatory motifs in functionally related sets of genes can be used in two ways. Given a gene set, an associated motif can be found by searching the upstream intergenic regions for short patterns occurring at an unusual frequency. Alternatively, given a novel motif whose function is unknown, an associated gene set can be found by testing a number of previously defined gene sets (*categories*) for enrichment.

In a single genome, motifs occur frequently by chance, and hence the enrichment observed is sometimes not sufficient to perform either of these two tasks with high sensitivity and specificity. With multiple aligned genomes at hand, most spurious motif instances can be eliminated and the enrichment should become more pronounced. We can use this increased power to assign a candidate function to the motifs discovered in the previous chapter and to discover additional motifs in a category-specific way.

In this chapter, we present methods to distinguish biologically meaningful motif instances under selective pressure from non-functional motif instances. We assign candidate functions to the genome-wide motifs discovered in the previous chapter and find that the majority of discovered motifs show a significant functional enrichment. We also present a new method to discover additional regulatory motifs associated with functional categories. For known factors, we find that our category-based discovery method has great sensitivity and specificity, finding concise binding sites even when previous methods fail. For all 354 categories tested, we find that only a small number of motifs are found and these are shared, reused across categories.

## 4.2. Constructing functionally-related gene sets.

In yeast, a number of genome-wide experiments have resulted in functional groupings of genes into *gene sets*. These represent possibly co-regulated groups, constructed from gene expression, transcription factor binding and protein function.

*Microarray technology* enables the simultaneous measurement of gene expression levels for all 6000 annotated yeast genes on a single array. Such arrays contain thousands of spots (one for every gene), each containing multiple single-stranded nucleotide probes complementary to the corresponding predicted yeast gene. When cell extract is washed on the array, the single-stranded mRNA transcripts present in the cell *hybridize* (bind) by complementarity to the appropriate spots in the array. The level of hybridization can be measured by first fluorescently labeling the mRNA transcripts and then measuring the level of fluorescence on each spot using a laser scanner. The higher the hybridization measured at a spot, the higher is the inferred level of mRNA expression for that gene. These genome-wide experiments have been repeated for hundreds of experimental conditions and expression profiles have been constructed for every gene, describing its expression levels in each condition. These profiles can then be clustered computationally<sup>59</sup>, typically by their pairwise correlation coefficients, to obtain sets of transcriptionally coordinated sets of genes.

Another technology, *ChIP*, has recently been applied to the genome-wide level to observe the binding locations of a transcription factor across the genome<sup>60,61</sup>. This technology enables the specific targeting of a transcription factor of interest, in order to pull it out of a cell extract. Pulling a transcription factor also selects for the DNA fragments that it is bound to. A researcher can then hybridize these fragments against an array containing probes for promoter regions, and infer which regions are bound by the transcription factor. Current technologies target transcription factors by either constructing an antibody specific to the factor, or by appending to the transcription factor a tag to which an antibody already exists (antibodies are molecules used by our immune system to recognize specific proteins of invading agents like viruses or bacteria; hence the name of Chromatin Immuno-Precipitation abbreviated as ChIP, referring to the use of antibodies to cause the chromatin bound by a factor to precipitate with the factor when

this one is pulled). The DNA is fragmented before precipitation and only a few hundred bases surrounding the bound site are typically pulled.

Genes can also be grouped into *functional categories*, based on the experimentally determined function of the proteins they encode. The function of thousands of yeast genes has been experimentally determined (to various degrees of precision). The scientific papers that describe these functions have been manually curated by the Saccharomyces Genome Database (SGD) group, generating a vast repository of knowledge. This knowledge has been classified hierarchically into Gene Ontology (GO) information or MIPS<sup>62</sup>, using a unified language that crosscuts species and organism boundaries. This hierarchy groups at each internal node genes of related function, from the most specific to the most general, in categories such as ‘meiotic DNA double-strand break processing’, ‘cell cycle’, or ‘metabolism’. Genes of related function will sometimes be part of the same metabolic pathway, required simultaneously for the correct sequence of chemical modifications of a metabolite, and hence likely to be co-regulated. Similarly, proteins that are part of the same protein complex are likely to be co-regulated, since they are required simultaneously for the correct assembly of the protein complex. Experimental methods similar to ChIP can be used to detect protein complexes<sup>63</sup>: an antibody specific to one of the proteins in the complex is used to pull the entire complex out of cell extract; the complex pulled is then fractionated at specific residues and the charge/weight combination of the fragments obtained by Mass Spectroscopy are used to find the precise set of amino acids in the fragment and the corresponding proteins that can result in such amino acid subsets.

#### **4.3. Assigning a function to the genome-wide motifs**

We used the biological knowledge captured in these sets of functionally related genes to assign function to the 72 genome-wide motifs discovered in the previous chapter. Since motifs can be degenerate and sometimes conserved in only a subset of the species, we first developed methods to score conserved motif instances. We then evaluated the overlap between the set of intergenic regions with motif scores above a given cutoff, and each functionally-related set of genes. We found a strong overlap with functional sets for most of the genome-wide motifs, and discover novel motif functions.



We used a probabilistic representation to detect conserved motif instances. We interpret every genome-wide motif  $m$  of length  $L$  as a probabilistic model, generator of sequences of length  $L$  over the alphabet  $\{A,C,G,T\}$ . We then evaluated for every genome position, the probability that the sequence was generated by motif  $m$ , and compared this to the probability that the sequence was generated at random, given the ratio of A,C,G,T in the genome. We evaluated each species in turn, to obtain a total number of bits in the alignment. Since gaps may exist in the alignment, we did not evaluate the motif match directly on the alignment. Instead, we evaluated the motif in the ungapped sequence of each species in turn, and translated the motif start coordinates based on the alignment. To avoid evaluating each of 12 million start positions in the yeast genome against the motif, we first hashed the four genomes for rapid lookup, and subsequently only search those intergenic regions that contain k-mers in the motif searched. To allow for degenerate matches, we also search for k-mers with one or two degeneracies from the query motif. We then used a simple threshold  $t$  and obtain the list of all intergenic regions containing conserved instances of the motif with score at least  $t$ . These instances are either upstream of downstream of each flanking gene, depending on its transcriptional orientation. We could thus generate an ‘upstream’ list of genes that contain these conserved instances in their upstream regions, and a corresponding ‘downstream’ list of genes. We compared the overlap between each upstream and downstream gene list against each set of functionally related genes.

We did not expect a perfect overlap where every gene in a category would contain the motif and every gene outside the category would not contain the motif. On one hand, we expected discrepancies due to experimental errors, incomplete annotations and artifacts of the clustering algorithms. But even with perfect data, discrepancies arise from molecular processes that cross-cut functional categories, transcription factor binding that is dependent on additional protein-protein interactions or chromatin structure, expression clusters that are controlled by multiple transcription factors. At the same time, much like spurious motif instances can occur in a single species when motifs are short and degenerate, even conserved motif instances can occur by chance, although less frequent. Similarly, functional motif instances may appear diverged due to alignment errors, or may have genuinely diverged across the species compared.

Thus, we evaluated the overlap between motif presence and functional information probabilistically. Assume that  $m$  genes contain the motif and  $r$  genes belong to a particular functional category. At random, if the motif is independent from the category, we expect the same proportion of genes to contain motif instances both inside and outside the category. The probability of observing a deviation from that ratio can be evaluated using the hypergeometric distribution, described in the appendix. If  $k$  genes are observed in the overlap between the two sets, and  $n$  genes are present in the yeast genome, we calculate a P-value that the enrichment is observed at random as the hypergeometric sum for all values of  $k'$  that are greater or equal to  $k$ . Since we were evaluating the overlap of each motif against a large number of candidate functional categories, we use a Bonferroni correction for multiple hypothesis testing.

We applied these ideas to the motifs we discovered in our genome-wide search. As a control, we used the Gal4 motif (Figure 4.1). Given the biological role of Gal4, we considered the set of genes annotated to be involved in carbohydrate metabolism (126 genes according to the Gene Ontology (GO)<sup>64</sup> classification) with the set of genes that

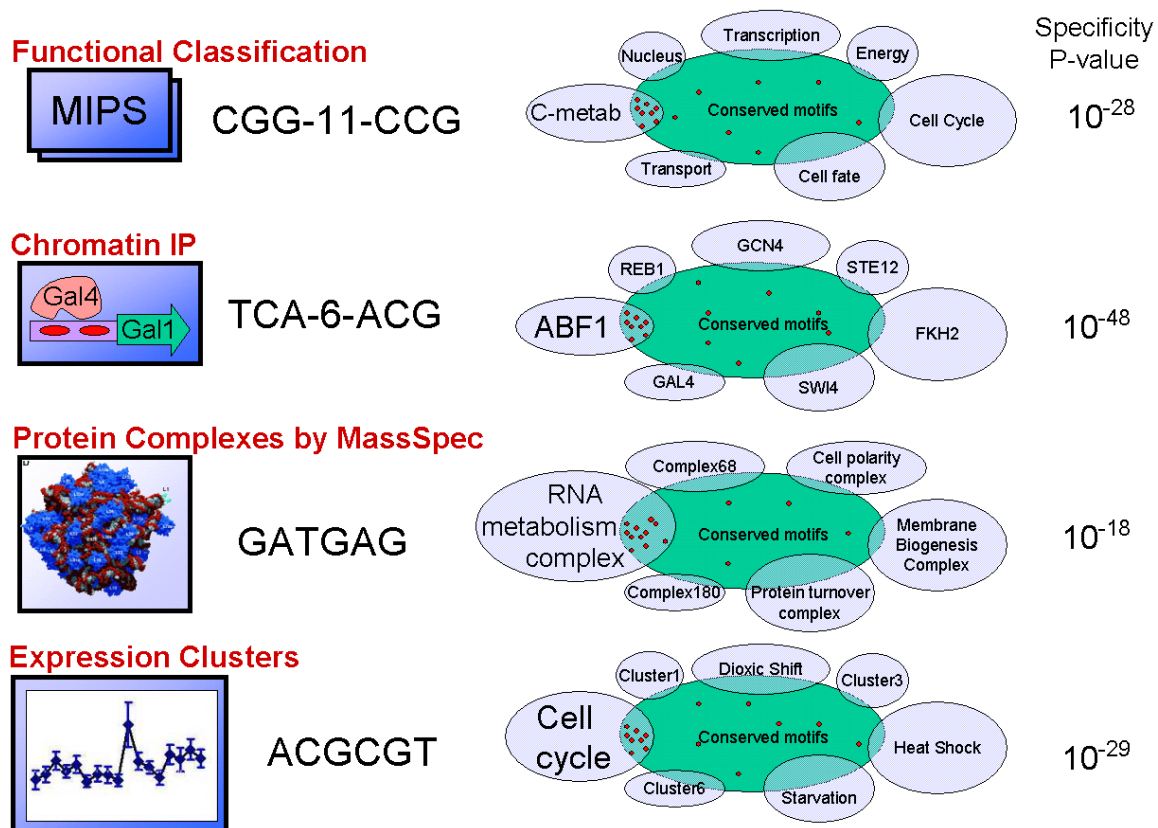


Figure 4.1. Assigning functions to genome-wide motifs based on functionally-related gene sets.

have a Gal4 binding motif upstream. The intergenic regions adjacent to carbohydrate metabolism genes comprise only 2% of all intergenic regions, but 7% of the occurrences of the Gal4 motif in *S. cerevisiae* (3.5-fold enrichment) and 29% of the conserved occurrences across the four species (15-fold enrichment). These results suggest that a function of the Gal4 motif could be inferred from the function of the genes adjacent to its conserved occurrences. Such putative functional assignments can be useful in directing experimentation for understanding the precise function of a motif.

### **Novel functions for genome-wide motifs**

We compared each of the 72 motifs against a collection of 318 yeast gene categories based on functional and experimental data described earlier. These categories consist of 120 sets of genes defined with a common GO classification in SGD<sup>64</sup>; 106 sets of genes whose upstream region was identified as binding a given transcription factor in genome-wide chromatin immunoprecipitation (ChIP) experiments<sup>61</sup>; and 92 sets of genes showing coordinate regulation in RNA expression studies<sup>59</sup>. To measure how strongly the conserved occurrences correlated with the regions upstream (or downstream) of a particular gene category. We require a hypergeometric score of at least  $10^{-5}$  to judge an overlap as significant, after accounting for testing of multiple categories. Most of the 36 discovered motifs that correspond to known motifs showed strong category correlation. Categories with the strongest correlation included those identified by ChIP with the transcription factor known to bind the motif, although many other relevant categories were identified. Of the 42 novel motifs, 25 show strong correlation with at least one category and thus can be assigned a suggestive biological function (Table 3.6).

Some motifs appear to define previously unknown binding sites associated with known transcription factors. Motif 32 is likely to be the binding site for Rgt1, which regulates genes involved in glucose transport<sup>65</sup>; the motif occurs upstream of many such genes, including appearing five times upstream of HXT1, which encodes a high-affinity glucose transporter. Motifs 21, 31 and 51 are all associated with genes whose upstream regions are bound by Sum1, a transcriptional repressor of genes involved in meiosis. The first motif has been previously reported (MSE)<sup>66</sup>, but the latter two are novel and occur near genes whose products are involved in chromatin silencing and transcriptional repression.

Other motifs do not match regions bound by known transcription factors, but show strong correlation with functional categories. Motif 9 occurs upstream of genes involved in nitrogen metabolism, including amino acid and urea metabolism, nitrogen transport, glutamine metabolism and carbamoyl phosphate synthesis. Motif 25 is enriched among co-expressed genes (expression cluster 37) whose products function in vesicular traffic and secretion, including GDP/GTP exchange factors essential for the secretory machinery, clathrin assembly factors and many vesicle and plasma membrane proteins. Motifs 9, 13, 26, 34, 37 may play a role in filamentation. They are all enriched in genes co-regulated during environmental changes, involved in signaling and budding and bound by transcription factors involved in filamentation, such as Phd1.

Six motifs show higher conservation downstream of ORFs. Some of these may be in the 3' untranslated region of a transcript and play a regulatory role in mRNA localization or stability. The strongest (Motif 6 and <sup>67</sup>) is found at genes whose product localizes to the cytosolic translational machinery, the mtDNA translational machinery or the mitochondrial outer membrane. Downstream motifs are also found enriched in a group of genes repressed during environmental stress (Motif 60 with expression cluster 37) and a group of genes involved in energy production (Motif 66 with expression cluster 46).

Two motifs (Motif 11a and Motif 69) show variable gap spacing, suggesting a new type of degeneracy within the recognition site for a transcription factor complex. Motif 11a corresponds closely to the known motif for Swi4 (Motif 11) but is interrupted by a central gap of 5, 7 or 9 bases; these variant motifs all show strong correlation with genes bound by Swi4 in ChIP experiments.

#### **4.4. Discovering additional motifs based on gene sets**

We next explored whether additional motifs could be found by searching specifically for conservation within individual gene categories. We selected mini-motifs based on their enrichment in specific categories and extended them to full motifs. We first evaluated our motif discovery method for ChIP experiments of factors with known motifs, and we found high sensitivity and specificity. We then searched for novel motifs in all 318 functional categories and discovered novel motifs.

The enrichment of regulatory motifs found in co-regulated gene sets has been the primary motivation for motif discovery algorithms such as MEME, AlignAce or BioProspector. These algorithms typically search for frequently occurring motifs within the set and subsequently evaluate the significance of the enrichment observed based on the overall frequency of the motifs throughout the genome. Thus, they search for motifs that are frequent within the set, and filter out those that are also frequent outside the set. We select for both criteria simultaneously by choosing mini-motifs based directly on their category enrichment score. We counted the conserved instances within the category (IN), and the conserved instances outside the category (OUT). We estimated the ratio  $p=IN/(IN+OUT)$  that we should expect for the category, based on the entire population of mini-motifs. We then calculated the significance of an observed enrichment as the binomial probability of observing IN successes out of IN+OUT trials given the probability of success  $p$ . We assigned a z-score to each mini-motif, as described in the genome-wide search. We extended those mini-motifs of z-score at least 5 sigma by searching for neighboring conserved bases that increase the specificity. We finally collapsed motifs of similar extension based on sequence similarity.

Factor	Known Motif	Hyper	MEME motif (Lee et al)		Category-based motif		Comparison
Abf1	RTCRYnnnnnACG	91.4	TRTCAYT-Y--ACGRA	good	RTCACnnnnnACGA	good	same
Gcn4	ATGACTCAT	47.8	TGAGTCAY	good	RTGACTCA	good	same
Reb1	CCGGGTAA	44.7	SCGGGTAA	good	CCGGGTAA	good	same
Mcm1	TTWCCcnwwwrGGAAA	35.9	TTTCC-AAW-RGGAAA	good	TCCnnnnnnGGA	good	same
Rap1	ACACCCATACATTT	30.0	TTWACAYCCRTACAY-Y	good	ACCCCA.ACA	good	same
Cbf1	RTCACRTG	24.2	TRGTCACGTG	good	GTCACGTG	good	same
Fkh2	TTGTTTACST	20.7	TTGTTTAC-TWTT	good	TGTTTTAC..TT	good	same
Swi4	CRCGAAAA	19.9	CSMRRCGCGAAAA	good	CAACRCGAAAA	good	same
Mbp1	ACGCGT	19.6	G-RR-A-ACGCGT-R		AACGCGTCG	good	better (+)
Ste12	RTGAAACA	17.8	GSAASRR-TGATRAWGYA		YTGAAACA	good	better (+)
Gal4	CGGnnnnnnnnnnCCG	16.1	CGGM--CW-Y--CCCG		CGGnnnnnnnnnnCCGA	good	better (+)
Swi6	ACGCGT	15.6	WCGCGTCGCGTY-C	good	ACGCGT	good	same
Pho4	CACGTG	14.2	TTGTACACTTYGTTT		CGCACGTG	good	better (+)
Hsf1	TTCTAGAA	14.1	TYTTCYAGAA--TTCY	good	GTTCTAGAAAnnTTCnnG	good	same
Dig1	RTGAAACA	13.6	CCYTG-AYTTCW-CTTC		TGAAACR	good	better (+)
Ino4	CATGTGAAat	13.4	G..GCATGTGAAAA	good	G...CATGTGAA	good	same
Fkh1	TTGTTTACST	13.2	CYTRTTTAY-WTT	good	TGTTTAC	good	same
Leu3	CCGGNNCCGG	13.1	GCCGGTMMCGSYC-	good	CCGGnnnnCGG	good	same
Bas1	TGACTC	10.2	CS-CCAATGK-CS		TGACTCTA	good	better (+)
Swi5	KGCTGR	9.2	CACACACACACACACA		TGCTGG	good	better (+)
Hap4	TnRTTGGT	8.5	YCT-ATTSG-C-GS		TGATTGGT	good	better (+)
Rlm1	CTAWWWWTAG	8.4	A-CTSGAAGAAATGCGGT		CTA..TTTAG	good	better (+)
Ino2	CATGTGAAat	7.4	GCATGTGAAAA	good	CATGTG	good	same
Met31	AAACTGTGGC	7.0	GCACGTGATS		TGTGGC	good	same
Ace2	GCTGGT	5.2	GTGTGTGTGTGTGTG		TGCTGGT	good	better (+)

**Table 4.2. Category-based motif discovery shows increased power to discover concise motifs.**

Hyper shows the enrichment of the previously published motif in the ChIP experiment corresponding to the factor. For slightly enriched motifs, MEME fails to find the correct motif, but the conservation-based method succeeds. Concise and correct motifs are found in each case.

We first evaluated our ability to detect the 43 known motifs for which ChIP experiments<sup>61</sup> had been performed with the transcription factor that binds the motifs. For each category defined by the ChIP experiment, we undertook category-based motif discovery. Strong category-based motifs were found in 29 cases and these invariably corresponded closely to the known motifs (Table 4.2). These include 11 cases in which the motif had not been found by genome-wide motif discovery, suggesting that a category-based approach can be more sensitive in some cases. No strong category-based motifs were found for the remaining 14 known cases, including 7 cases in which genome-wide analysis yielded the known motif. Analysis of these 14 known motifs showed that none were, in fact, enriched in the ChIP-based category. This may reflect errors in the known motifs in some cases and imperfect ChIP data in others. Genome-wide analysis may simply be more powerful than category-based analysis in some instances. In all, 46 of the 55 known motifs were found by either genome-wide or category-based analysis. The remaining 9 cases may reflect true failures of the comparative genomic analysis or errors in the known motifs.

We compared our results to the motifs discovered by MEME in a single species as reported in Lee et al<sup>61</sup>. Our method showed stronger sensitivity in discovering all motifs for which the ChIP experiment indeed contained the correct motif. Additionally, the method showed strong specificity in the motifs discovered: the motifs were short and concise, and closely matched the published consensus. On the contrary, MEME failed to find the true motif in a number of cases, and when a motif was found it was generally obscured by a number of surrounding spurious bases that are not reported in the known motifs. Thus, we successfully used the additional information that comes from the multiple alignment to improve category-based motif discovery with very satisfactory results. By comparing multiple species, the signal becomes stronger. It allows the search to focus on the conserved bases, eliminating most of the noise. Table 1 summarizes the results. For each factor, we show the published motif, the hypergeometric enrichment score of the motif within the category (Hyper), the motif discovered by MEME and a quality assessment, the motif discovered by our method, as well as the corresponding category-based score and a quality assessment, and finally the comparison of our method to MEME. The performance of MEME degrades for less enriched motifs, but we consistently find the correct motif.

Table 4 **Additional new motifs discovered by category-based analysis**

No.	Category	Category-based motif	Interpretation	Score
1	Exp.: cluster 37	YCCCTTAAA	New: cluster 37 (Msn2/4-like)	[8.5]
2	ChIP: <i>FHL1</i> in YPD	ATGTACGGATG	New: Rap1 alternate	[7.6]
3	GO: carbohydrate transport	GTTTTTCCG	New: carbohydrate transport	[7.2]
4	GO: fatty acid beta-oxidation	TTAnnnCCG	New: fatty acid oxidation	[6.3]
5	GO: glycolysis/glyconeogenesis	TAGTGGAAGC	New: glycolysis/glycogenesis	[6.0]
6	Exp.: cluster 37	TCAGCC	New: cluster 37	[5.9]
7	Exp.: cluster 37	CGGnnnnCGG	New: cluster 37	[5.7]
8	ChIP: <i>CIN5</i> in YPD	GnTTAnnTnAGC	New: Cin5 alternate	[5.6]
9	ChIP: <i>STE12</i> in butanol	CATTCT	Known: Tec1	[5.4]

**Table 4.3. Novel category-based motifs.**

We then applied the approach to all 318 gene categories. A total of 181 well-conserved motifs were identified, with many of these being equivalent motifs arising from multiple categories. Merging such motifs resulted in 52 distinct motifs, of which 43 were already found by the analyses described above. The remaining 9 motifs represent new category-based motifs (Table 4.3), including the following.

Three novel motifs are associated with genes that are bound by the transcription factors Rap1, Ste12 and Cin5, respectively. Rap1 is known to bind incomplete or degenerate instances of the published motif and the new motif may confer additional specificity. The motif associated with Ste12 is the known binding site for the partner transcription factor Tec1, suggesting that Ste12 binding is strongly associated with its partner under the conditions examined. Similarly, the novel motif associated with Cin5 may be that of a partner transcription factor. Three novel motifs are associated with the GO category for carbohydrate transport, fatty-acid oxidation and glycolysis-glycogenesis, respectively. Three novel motifs are associated with an expression cluster (cluster 37) that includes many genes involved in energy metabolism and stress response.

#### 4.7. Conclusion

Category-based motif discovery contributes only a modest number of additional motifs beyond those found by genome-wide analysis. This confirms the relatively small number of regulatory motifs in yeast. A limited count is surprising given the large number of coordinately transcribed processes in yeast. The versatility of fine-grain yeast regulation may be rooted in a combinatorial control of gene expression, which will be the topic of the next chapter.

## CHAPTER 5: COMBINATORIAL REGULATION

### 5.1. Introduction

We also used the comparisons to understand combinatorial interactions between regulatory motifs. A simple view of gene regulation where each environmental response is regulated by a dedicated transcription factor would require as many transcription factors and regulatory motifs as there are molecules and environmental changes. This is however not the case. It is estimated that only 160 transcription factors exist in the yeast genome, but yeast cells contain thousand of co-regulated sets of genes. This discrepancy requires a different model of gene regulation that goes beyond a one-to-one correspondence between regulatory motifs and cellular processes.

Our results from the previous chapter indeed point to a model where specific motif combinations are responsible for different cell responses. We saw that a single motif is typically involved in the control of many processes, and that a single process is typically enriched in multiple regulatory motifs. Furthermore, we saw that different processes were enriched in different combinations of regulatory motifs. Protein-protein interactions between the multiple factors bound upstream of every gene may dictate the specific combination of conditions under which the gene will be expressed. Understanding the combinations of regulatory motifs that are biologically meaningful, and the changing target gene sets may explain the versatility of eukaryotic gene regulation using only a small number of regulatory building blocks.

In this chapter, we develop methods to reveal the combinatorial control of gene expression. We construct a global motif interaction map, simply based on proximity of conserved motif pairs without requiring biological knowledge of gene function. We then present evidence for the changing functional specificities of the motif combinations discovered. Finally, we show the genome-wide effect of motif combinations on gene expression change.

### 5.2. Motifs are shared, reused across functional categories

We saw in the previous chapter that the motifs discovered across different categories largely overlapped. Each motif was discovered on average in three different



categories. This overlap is certainly to be expected between functionally related categories such as the chromatin IP experiment for Gcn4, the expression cluster of genes involved in amino acid biosynthesis, as well as the GO annotations for amino acid biosynthesis, all of which are enriched in the Gcn4 motif, the master regulator of amino acid metabolism.

More surprisingly however, different transcription factors are often enriched in the same motif (which may be due to cooperative binding), and the same motif appears enriched in multiple expression clusters and functional categories. For example, Cbf1, Met4, and Met31 share a motif, and so do Hsf1, Msn2 and Msn4; Fkh1 and Fkh2; Fhl1 and Rap1; Ste12 and Dig1; Swi5 and Ace2; Swi6, Swi4, Ash1 and Mbp1. Also, a single motif involved in environmental stress response is found repeatedly in numerous expression clusters, and in functional categories ranging from secretion, cell organization and biogenesis, transcription, ribosome biogenesis and rRNA processing.

Hence, the set of regulatory motifs that are specific to one functional category seems limited. This can hamper category-based motif discovery methods: no category will be enriched in a single motif, and no motif will be enriched in a single category. Additionally, there are a number of experimental limitations to a category-based approach. For example, the expression clusters we have used, although constructed over an impressive array of experiments, are still limited to the relatively few experimental conditions generated in the lab. Additionally, the functional categories we used are limited to the few well-characterized processes in yeast, and the molecular function of more than 3000 ORFs remains unknown.

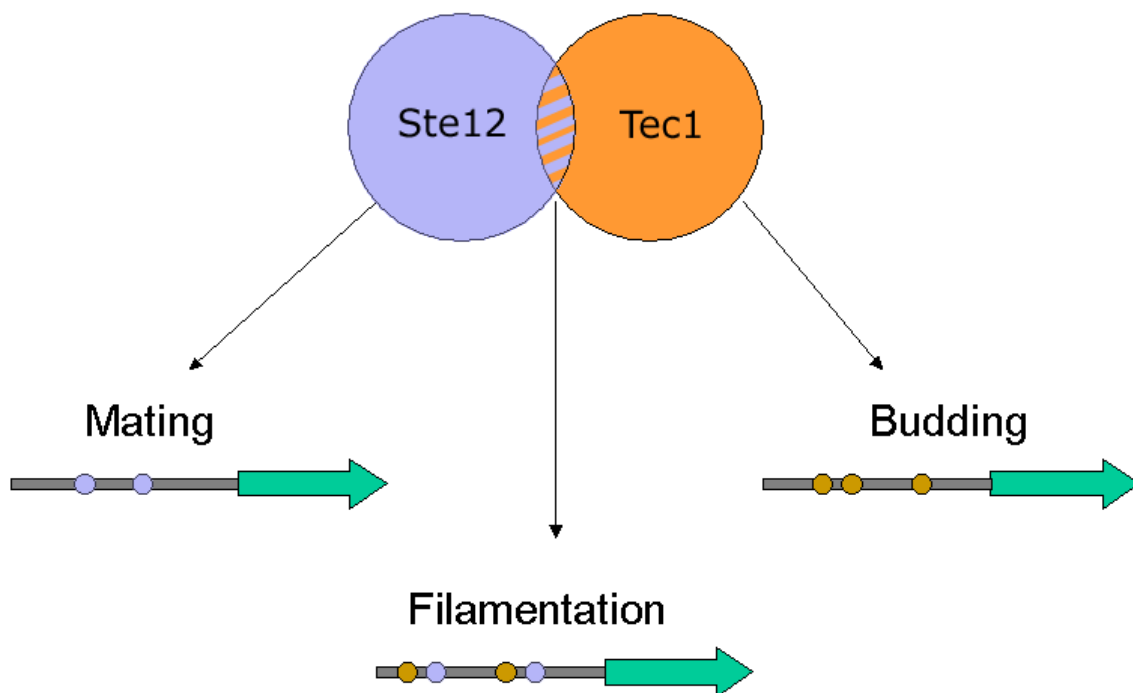
A genome-wide approach presents a new and powerful paradigm to understanding the dictionary of regulatory motifs. By discovering in an unbiased way the complete set of conserved sequence elements, we now have the building blocks to subsequent analyses of regulation. To understand the full versatility of gene regulation, we now turn to understanding the combinatorial code of motif interactions. We first show that motif combinations can change the specificity of target genes, not in an additive, but in a combinatorial way. We then present methods to discover interacting motifs from the

genome-wide co-occurrence of their conserved instances, without making use of functional information. We then show that the interactions found are meaningful.

### 5.3. Changing specificity of motif combinations.

The effect of motif sharing a reuse can be additive or combinatorial. An additive effect simply adds the effect of the co-occurring transcription factors. For example, if each of two factors induces the expression of a gene, and both bind to a particular region, then their effect would be a doubly increased level of transcription for that gene. A combinatorial effect can be more complex. Namely, the combination of two factors may repress expression for a gene, even though either of the factors alone induces its expression.

Similarly, we should find that transcription factor combinations show different functional specificities than either of the transcription factors alone (Figure 5.1). We study here the gene category enrichment of two transcription factors that are known to bind to DNA cooperatively: Ste12 and Tec1. We considered three types of regions: those containing Tec1 motifs but no Ste12 motifs, those containing Ste12 motifs but no Tec1 motifs, and those containing both Ste12 and Tec1 motifs. We then intersected these



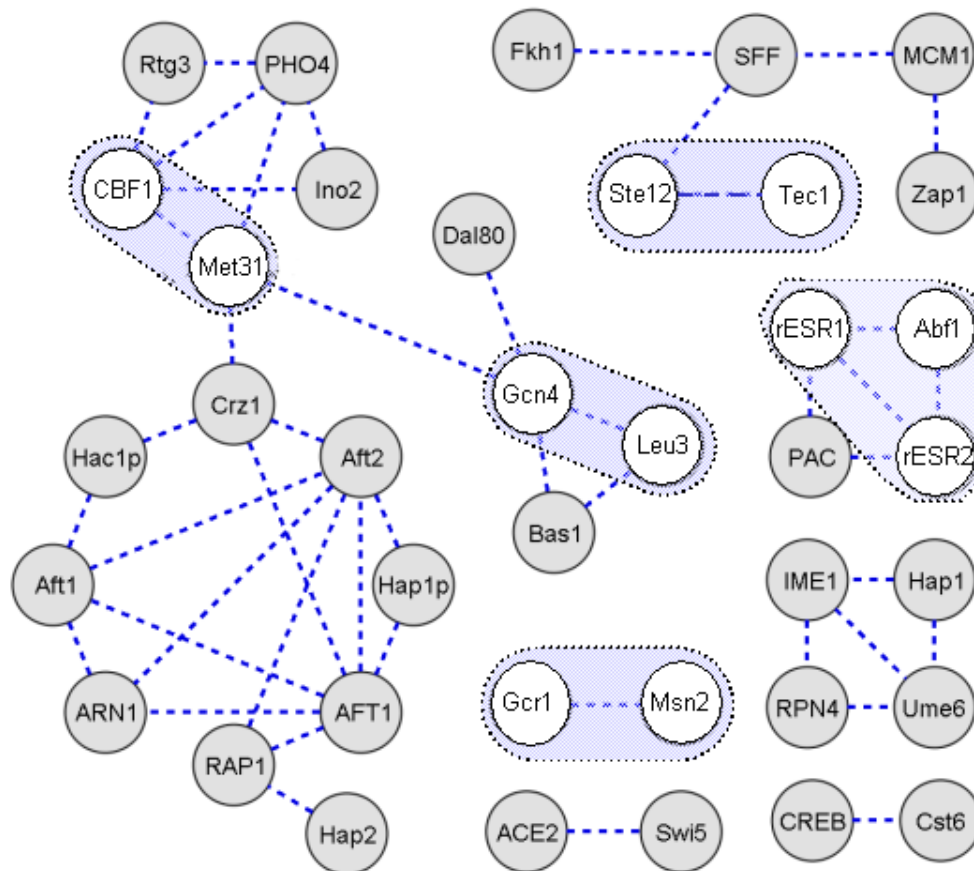
**Figure 5.1. Changing specificity of motif combinations increases versatility of gene regulation.**

three types of regions against the gene sets described previously.

We found that the regions that contain only the conserved Ste12 motif are enriched for genes involved in mating and pheromone response, while those that contain conserved occurrences of both the Ste12 and Tec1 motifs are enriched for genes involved in filamentous growth. These computational observations are consistent with recent elegant work showing genome-wide evidence that Ste12 and Tec1 indeed cooperate during starvation to induce filamentation-specific genes<sup>68</sup>. We also found that regions that contain only conserved occurrences of the Tec1 motif are enriched for genes involved in budding and cell polarity, suggesting that Tec1 has functions that do not require cooperative binding with Ste12.

#### 5.4. Genome-wide motif co-occurrence map.

We next address the question of discovering these motif interactions in a genome-



**Figure 5.2. Genome-wide motif co-occurrence map** reveals biologically meaningful motif relationships and transcription factor interactions

wide fashion. Protein-protein interactions between cooperatively binding transcription factors require that they bind in proximity upstream of their target genes. The regulatory motifs recognized by these factors should therefore co-occur in these intergenic regions of cooperative binding. The spatial orientation and physical distance between these motifs may vary across different genes, the varying distances being compensated by DNA bending that can bring the two sites in proximity. However, motif interactions do not typically cross gene boundaries, that are enforced by chromatin packaging and larger physical distances from one intergenic region to the next. Thus, co-occurrence of regulatory motifs in the same intergenic regions might be a good indicator of interacting transcription factors.

Using the comparison of the four species, we observed the genome-wide co-occurrence patterns of regulatory motifs (Figure 5.2). We searched for motifs that occur in the same intergenic regions more frequently than one would expect by chance. We computed the probability of seeing at least  $k$  regions in common when one motif is found in  $m$  regions and the other motif is found in  $r$  regions, given a total of  $n$  intergenic regions using the hypergeometric distribution.

Without using any functional information of gene categories, we found a number of significant motif interactions. These group motifs together into complex motif co-occurrence networks that may form the basis for studying combinatorial regulation of gene expression. These are not apparent in a single genome, where functional instances of the motif are overwhelmed by a much larger number of random occurrences. Cross species conservation greatly decrease this random noise and reveals biologically meaningful correlations.

## **5.5. Results.**

We outlined here a number of biologically significant connections in the motif co-occurrence map. The combinatorial effect between Ste12 and Tec1 was indeed observed at the genome-wide level. The Ste12 and Tec1 motifs show clear correlation, with about 20% of regions having a conserved occurrence of one also having a conserved occurrence of the other. This enrichment is not apparent when considering *S. cerevisiae* alone.

The motif co-occurrence map reveals a number of biologically meaningful interactions. (a) About 60% of regions containing conserved motifs for the transcription factor Leu3 (which regulates branched-chain amino-acid biosynthesis) also contain conserved motifs for Gcn4 (a general factor regulating amino acid biosynthesis, as well as many other processes). (b) About 46% of regions containing conserved motifs for the transcription factor Met31 also contain conserved occurrences of Cbf1. In fact, Cbf1 (which is involved in DNA bending) is known to physically interact and cooperate with the MET regulatory complex. (c) About 34% of regions containing a conserved Gal4 motif also contain a conserved Mig1 motif. In this case, the correlation reflects antagonistic interaction. Gal4 induces galactose metabolism genes in presence of galactose, but Mig1 represses galactose metabolism in presence of glucose. (d) Pairwise co-occurrence connects a group of five motifs: Msn2/4 (general stress response), Rlm1 (response to cell-wall stresses), Pdr1 (pleiotropic drug resistance), Tea1 (Ty element activator) and Tbf1 (Telomere-binding factor). This suggests a possible link between various stress responses and adaptive changes at the genome level<sup>69</sup>.

Many additional correlations are seen among known and novel motifs and can be pursued experimentally and computationally to construct comprehensive co-occurrence networks. These can provide information valuable in deciphering biological pathways in yeast.

## **5.6. Conclusion.**

In this chapter, we provide methods to discover meaningful combinatorial interactions between regulatory motifs in a genome-wide way. Motif combinations can change the functional specificity of downstream motifs, and regulate a large number of processes using only a small number of regulatory motifs. This combinatorial nature of yeast regulation allows for a robust and modular regulatory network to adapt to changing environmental conditions. It is possible that additional regulatory motifs are added to the network, modulated by the more stable master regulatory motifs. We can further pursue these ideas to understand the rewiring of regulatory networks across evolutionary time. This may be one of many subtle ways of rapid evolutionary change outlined in the next chapter.

## CHAPTER 6: EVOLUTIONARY CHANGE

### 6.1. Introduction

In previous chapters, we used the stronger conservation of functional elements across related species for the direct identification of genes and regulatory motifs. However, the species compared are not identical. They live in different environments and are subject to different pressures for survival. In the short evolutionary time that separates them, they have undergone a number of evolutionary changes to adapt to their respective environments.

In comparative genomics, both similarities and differences of the species compared can reveal important biological principles. Focusing on the similarities gives us a view of a core cell whose functionality has remained unchanged since the common ancestor of the species. Focusing on the differences gives us a dynamic view of a changing genome, and the mechanisms evolved for rapid adaptation to changing environments.

In this chapter, we focus on the mechanisms of evolutionary change that have become apparent in our comparisons. We show that the ambiguities in gene correspondence found in chapter 1 are localized in rapidly evolving telomeric regions at the chromosome endpoints. We also show that non-telomeric changes in gene order are due to either the inversion of a chromosomal segment (containing fewer than 20 genes) or reciprocal exchanges of chromosomal arms. For both types of events, the sequences at the breakpoints suggest specific mechanisms of chromosomal change. We observed few differences in gene content between the species, suggesting that phenotypic differences may be due to more subtle effects like protein domain changes and changes in gene regulation. Finally, we observed rapidly and slowly evolving genes: at one end of the spectrum, we found evidence of positive selection for rapid change in membrane adhesion proteins, suggesting a small number of mechanisms of rapid change; we also found genes that were surprisingly strongly conserved suggesting new hypotheses for their function.

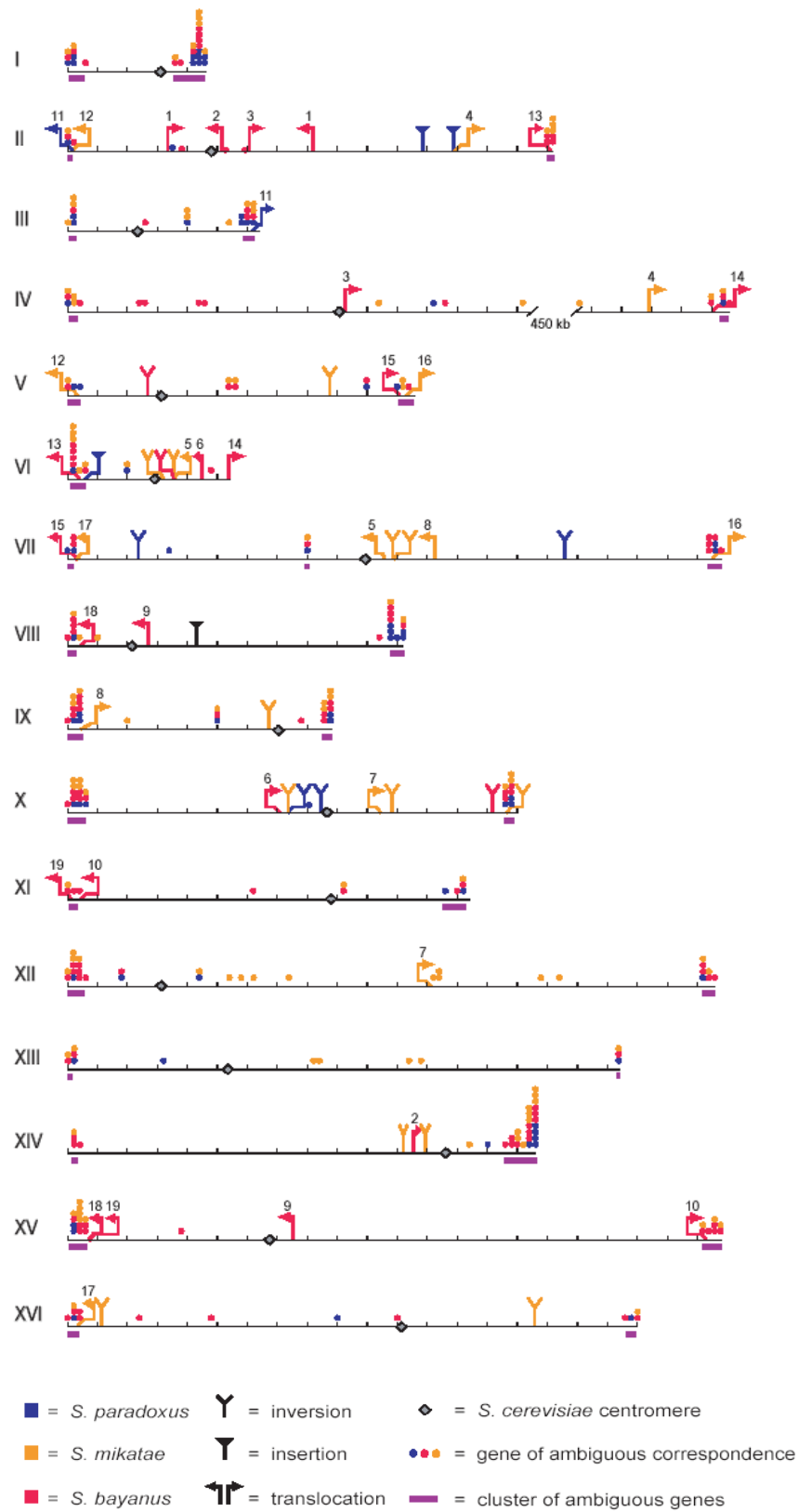
## 6.2. Protein family expansions localize at the telomeres.

In the previous chapters, we used unambiguous ORFs and intergenic regions to discover conserved coding and regulatory elements in the yeast genome. In this chapter, we use ORFs with ambiguous correspondence to determine regions of rapid change.

We marked the chromosomal location of all *S. cerevisiae* ORFs that are ambiguous in at least one species. We then constructed ambiguity clusters when two or more ambiguous ORFs within 16kb of each other. We counted the number of ambiguities in each cluster, counting more than one ambiguities for an ORF whose correspondence was ambiguous in more than one species. Only 32 clusters were found containing more than two ambiguities. We ignored two clusters due to regions of low coverage in *S. mikatae* and one cluster corresponding to a previously described inversion.

Most of the ambiguities are strikingly clustered in telomeric regions (Figure 6.1). More than 80% fall into one of 32 clusters of two or more genes (average size ~18 kb, together comprising ~4% of the genome), which correspond nearly perfectly to the 32 telomeric regions of the 16 chromosomes of *S. cerevisiae*. Only one telomeric region lacks a cluster and only one cluster does not lie in telomeric regions in *S. cerevisiae*: it is a recent insertion of a segment that is telomeric in the other three species. The rapid structural evolution in the telomeric regions can also be observed at the gene level. The gene families contained within these regions (including the HXT, FLO, PAU, COS, THI, YRF families) show significant changes in number, order, and orientation. The regions also harbor many novel sequences, including protein-coding sequences. Finally, the telomeric regions have undergone 11 reciprocal translocations across the species.

Together, these features define relatively clear boundaries for the telomeric regions on all 32 chromosome arms, with sizes ranging from ~7 kb to ~52 kb on chromosome I-R. The extraordinary genomic churning occurring in these regions - and the telomeric localization of environment adaptation protein families - together probably play a key role in rapidly creating phenotypic diversity over evolutionary time. A high degree of variation in telomeric gene families has also been reported in *P. falsiparum*<sup>69</sup>, the parasite responsible for malaria, and is related to antigenic variation.



**Figure 6.1. Rapid evolution in telomeres.** Telomeric protein family expansions can rapidly create phenotypic diversity, potentially an evolutionary advantage.



### 6.3. Chromosomal rearrangements mediated by specific sequences.

Outside of the telomeric regions, few genomic rearrangements are found relative to *S. cerevisiae* (Figure 6.2). To discover these, we considered consecutive unambiguous matches, marking all changes in gene spacing, gene orientation, and off-synteny matches between scaffolds and orthologous *S. cerevisiae* chromosomes. We found that changes in gene spacing are typically associated with transposon insertions and associated novel genes, as well as tandem duplications. Virtually all changes in gene orientation typically affect between 2 and 10 consecutive ORFs and can be traced to one of 16 multi-gene inversions. The majority of off-synteny matches involve a single ORF and only 20 involve more than 2 consecutive ORFs. Virtually all single-gene off-synteny matches were contained within ancient duplication blocks of *Saccharomyces* as described in <sup>70</sup> and <http://acer.gen.tcd.ie/~khwolfe/yeast/nova/>. These probably represent previously duplicated genes that were differentially lost in different species, rather than a DNA

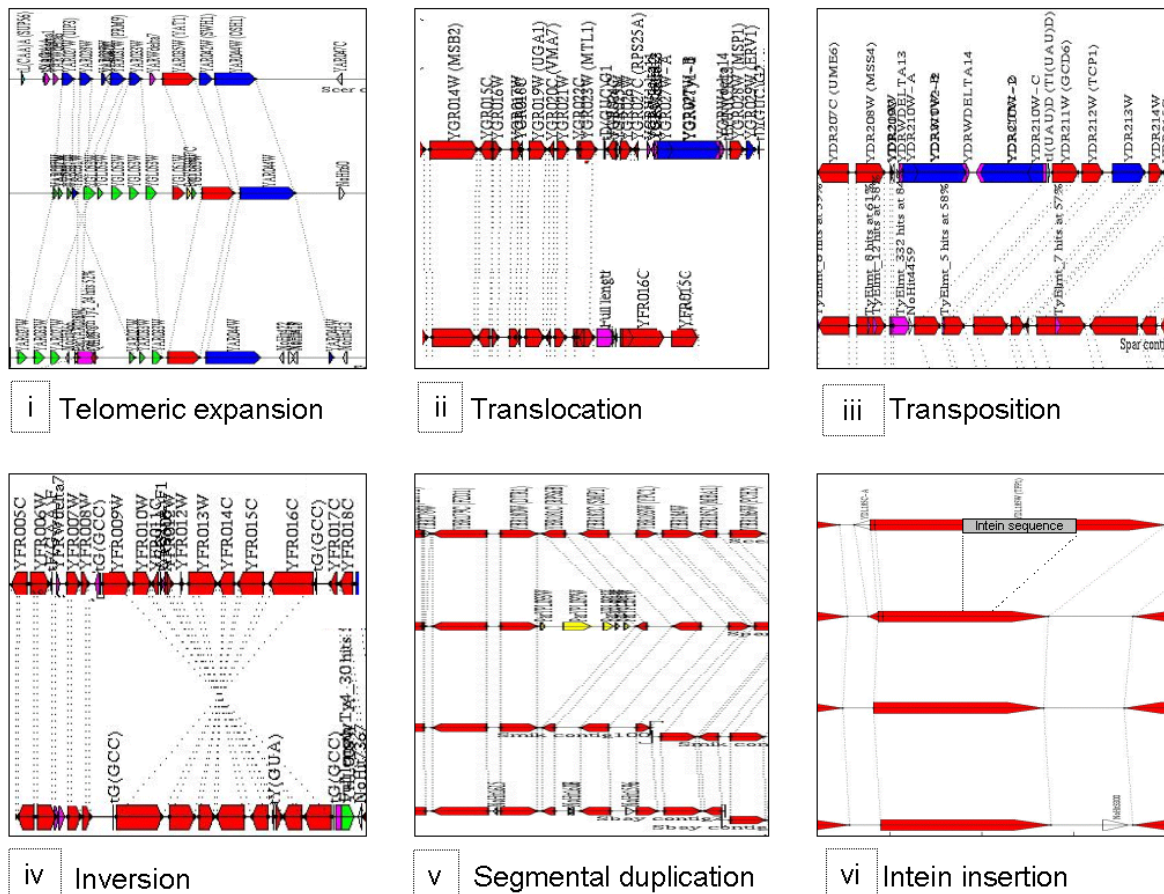


Figure 6.2. The six types of genome rearrangements that separate the species.

break in one of the two lineages, as was previously noted in <sup>71</sup>. Off-syteny matches that involve more than two genes from the same chromosome correspond to one of 20 chromosomal exchanges.

*S. paradoxus* shows no reciprocal translocations, 4 inversions and 3 segmental duplications. *S. mikatae* shows 4 reciprocal translocations and 13 inversions. *S. bayanus* has 5 reciprocal translocations and 3 inversions. The results confirmed four recently reported reciprocal translocations in these species, identified by pulsed-field gel electrophoresis<sup>72</sup>, and identified four additional reciprocal translocations that had been missed. The sequence at the chromosomal breakpoints suggested the possible mechanism that underlie the rearrangements. Strikingly, the 20 inversions are all flanked by tRNA genes in opposite transcriptional orientation and usually of the same isoacceptor type; the origins of inversions in recombination between tRNA genes has not previously been noted. The reciprocal translocations occurred between Ty elements in seven cases and between highly similar pairs of ribosomal protein genes in two cases; the implication of Ty elements in reciprocal translocation is consistent with previous reports<sup>44,71-73</sup>. One segmental duplication involves ‘donor’ and ‘recipient’ regions that are descendants of an ancient duplication in the yeast genome<sup>70</sup>. Differential gene loss of anciently duplicated genes has been previously reported<sup>74</sup>, but this is the first observation of a recent re-duplication event within anciently duplicated regions.

#### **6.4. Small number of novel genes separate the species**

We found a very small number of genes unique to one species and absent in the others. We noted above that *S. cerevisiae* contains 18 genes for which we could not identify orthologs in any of the other species, of which 7 encode  $\geq 200$  aa. These may be species-specific genes in *S. cerevisiae*, but alternatively could simply reflect gaps in the available draft genome sequences.

This uncertainty does not arise, however, in the reverse direction in identifying genes in the related species that lack an ortholog in *S. cerevisiae*. We found a total of 35 such ORFs encoding  $\geq 200$  aa (with the minimum length chosen to ensure that these are likely to represent valid genes). The list includes 5 genes unique to *S. paradoxus*, 8 genes unique to *S. mikatae* (two of which are 99% identical) and 19 genes unique to *S. bayanus*

(three of which form a gene family with  $\geq 90\%$  pairwise identity). There is also one gene represented by orthologous ORFs found in the latter two species only and one represented by orthologous ORFs in all three related species.

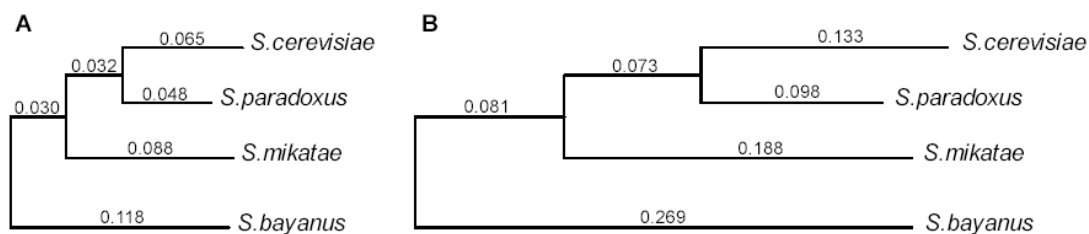
These species-specific ORFs are notable with respect to both function and location. The majority (63%) can also be assigned biological function on the basis of strong protein-sequence similarity with genes in other organisms. Most involve sugar metabolism and gene regulation (including one encoding a silencer protein). The majority (69%) are found in telomeric regions and an additional set (17%) are immediately adjacent to Ty elements; these locations are consistent with rapid genome evolution.

A curious coincidence was noted in the region between *YFL014W* and *YFL016W* in *S. cerevisiae*. In the orthologous regions in all four species, we find a species-specific ORF in every case (165, 111, 136 and 228 aa), but these four ORFs show little similarity at the protein level. The amino acid sequence has been disrupted by frame-shifting indels, but a long ORF has been maintained in each case. The explanation for this phenomenon is unclear, but may prove interesting.

### 6.5. Slow evolution suggests novel gene function.

With sequence alignments at millions of positions across the four species, it is possible to obtain a precise estimate of the rate of evolutionary change in the tree connecting the species.

One notable observation is the difference in substitution rate between *S. cerevisiae* and *S. paradoxus* (Figure 6.3). Using *S. bayanus* as an outgroup, the substitution rate is about 67% lower in the lineage leading to *S. paradoxus*. This observation is consistent regardless of the measure of evolutionary change: mutations,



**Figure 6.3. Slower mutation rate of *S. paradoxus* observed in genes and in intergenic regions**

insertions, deletions measured across intergenic regions, genes or degenerate nucleotides in coding sequence all point to the same discrepancy. Hence, we can conclude that *S. paradoxus* is evolving at a slower rate than *S. cerevisiae* or *S. mikatae*. This could be due to generation time, but also life cycle throughout the year. Wild-type species remain dormant most of the year in spores, until the next blooming. This causes fewer cell divisions, hence fewer errors in replicating the DNA.

We can also observe differences in the rate of change of individual genes. One case stands out as an extreme outlier: the mating-type gene MATA2. The gene shows perfect 100% conservation at the amino acid level over its entire length (119 aa) across all four species. More strikingly, the gene shows perfect 100% conservation at the nucleotide level as well (357 bp). This differs sharply for the typical pattern seen for protein-coding genes, which show relaxed constraint in third positions of codons. Notably, the MATA2 gene is the only one of the four mating-type genes (the others being MAT $\alpha$ 1, MAT $\alpha$ 2 and MATA1) whose biochemical function remains unknown despite two decades of research<sup>75</sup>. An important clue may be that the sequence of MATA2 is identical in all four species to the 3'-end of the MAT $\alpha$ 2 gene. Perfect conservation at the nucleotide-level and identity to the terminus of MAT $\alpha$ 2 suggests that MATA2 may function not only by encoding a protein, but by encoding an anti-sense RNA or a DNA site. Hence, the lack of evolutionary change can suggest additional biological functions responsible for the pressure to conserve nucleotide sequence.

#### **6.6. Evidence and mechanisms of rapid protein change.**

Similarly, the unusually high rate of change can be biologically meaningful. The gene analysis described in chapter 2 rejected only a single ORF (*YBR184W*) that is clearly encoding a functional protein. The region containing *YBR184W* corresponds to a large open reading frame in all four species (524, 558, 554 and 556 amino acids, respectively), but the alignment shows unusually low sequence conservation. The sequence has only 32% nucleotide identity and 13% amino acid identity across the four species (Figure 6.4). Pairwise alignments across the species show numerous insertions and deletions, explaining why the gene failed the RFC test. (Interestingly, multiple alignment of all four species simultaneously improves the alignment sufficiently that the gene passes the RFC test; this suggests a way to improve the test.)

The rapid divergence is suggestive of a gene under strong positive selection. We tested this notion by calculating the Ka/Ks ratio (the normalized ratio of amino-acid-altering substitutions to silent substitutions), a traditional test for positive selection<sup>76</sup>.

```

Scer MYQNNVLNAILASEKSNFQYD-SGTILRN-HKRPIITFNNNIEHTVSEPNNFTGYEEKED
Spar MDRNNVLNNISVSGKSNFQYEQNGKRRLKNQKRPIITFNSNAEYAI SEHEKYTNYEETTD
Smik MLNNISSSGNINVQYEQNING-RLKNDKTPTKSFNANVEYITCNYN SFESYEERVVLTNTM
Sbay MIPNNVLNDIADSGRLNVKYNQQIKVKLGNVKAQGI GLGANEMPP SENDFKYTSYEERIE
      *      *                               *

Scer L---DIMDICPYYP-----KARML--ADAIQHAKTSASENKMELSM-----KTI
Spar SIATNMCPYYRKGGILMDAPQSVMNQRANTSIGEDKKYVSEHN SGI LNPGNKVELSMKAT
Smik K---TCSNYQKGDILIEILQPVMNRTI-NTRISKSKKNPEYKSGVLSPEDAEESLMKKT
Sbay RQKIKTHYHYRKGGTRIDTSKVAINQHADTRIRGGRKCTPEHNI GTTTTSKHEARLPEDIP
                               *      *

Scer PCLKKENVHVEKGH DWSQLSTSRI CKI LEDI ADKKNKTRRQSAP LQKTKYFPTNENQNTD
Spar PCLKQEDCHFEGGH DWSQLSTSRI CKI LEDI SGKKHKTRGQLAP LQKVYPQKIGNQKTD
Smik KFLKRKSNHHKERH DWSRLSTSNICKI LEDI SGKKDRTRVQSSLLQEKIYPQKVCNQKIK
Sbay QPIEQENRHFEDKFDWSRLSTSKI CEI LEDI SSKKHRSKVHFTP LPKKEKLPKTHYKEND
      *      * * * * * * * * * *      *      *

Scer IENQNW SQIPNEDICALIEKIASRPNKNRKRKNLSCSKVQEI QGNIDLPPKDVQEGDISD
Spar KKNQNW SQIPNEDICALIEKISSRPNKNPKRINRSCSQIQEMLGGIDSAENRIDKGEITD
Smik ENQNW SQIPNEDICALIERIASRSPKLKRTNHPDYQVKEITDAIDAAGDCMRKVEGMQ
Sbay EQNQDWSQIPNEDVCELIEKIASRPNNSLKDMDRSCSRNQENSETIDFLENDTHIGELTN
      * * * * * * * * * *      *      * *

Scer SSLFAAVRGTKKVSGYDYNSEDKIPNAIRLPYCKQILRLFSLLQMKRNDLIVTSENCNSG
Spar SPLFTAVRENEDVLGYNFGSGKIPKAICLPHHKEIKQLVSFLQMKKNELGTTCKNHEGE
Smik TLLVKDEGSCENSRRRDFNSKSIIPNTISLPFQKDRKQLRSTLQQRKRS LVTISGTHHGK
Sbay TLQRNAKGSCEKT TENHYNPESKISKAICLPHQEKELR LIPFLQKQRREPAICRGGVRE
                               *      * *      * *

Scer VFFSNFNYQLQVKSNCIANI-----SSTLSFLPHHEITVYTSFILYPNVVDNIWECTRY-
Spar LVLRFKDDKVTVNSNCATNNYFNEVIATLNYSVYHEMTVYTSFILNPNVGDNIWGSRKCA
Smik FIMRELDNESIVKLNCVPVNRFFNQEKVILNHSFHHEIIVYTYTALQFKVENNIWKLKSP
Sbay FRMRKSPKKLKLCPRTLVRNVFSVGVIISSHSLSFERKILMHQTFELRSLRDIRDRRTSS
                               *      *

Scer -----AIQLLKSEAAQFTLLRDIYSGF TII LSNHRYHPKGF SADYCY SANELTLFLFVI
Spar FQLLKPEAVDTTNNMHHTLPQDIHGDSII VVSKYQFDPNNLVVELRYSSKKLRLLHSIF
Smik IQLLEPGVINTTNI LHPVSVPPQGLYGDFTVFLSKNLTPDKPKFIDGCCFSLQELRSLTCAF
Sbay FLRSKLEQAHT EITSHLFKPSQSI SPCFTMKVVKNR LGSKAFAIICHYELQTPQFDLRGP

Scer RTGQKKVLYRSIPH-----NTAAIEKDSSFDTENRKR RSEEEVVLKCRKCSNNSLALKE
Spar GTGSKKILYHLIPH-----NITTVKEDCPSDTEYSKKRSQKNVLK YRMYSSSL-VLKE
Smik GTYQKNELHHRIAY-----HTTTIEMDYSS TQYEKKKPKHNAIFRN RTHANSWLTPKK
Sbay KTSDCNTHKKSSRLLMPCEATAI KEIYHPDANLKS KTSHGNTVVETEKL FNNCLPRKGR
      *      *

Scer ISTYRLDSAEGFEKSQPLKDEAKLSDMNYVQGSISYNRTI LTGLWKLFHRLCCKDRYRKT
Spar ISAHGLDSVESFARSQSPENKRELS DINYVQGSVTHNRSILACLGDFH RFYFKSCSGKT
Smik VCVHRLDSAGCSHRFQPAEKKENHKDVNSLQGNDTRQRNIISDLRNFFLKFYCNGCSKKT
Sbay ADLLNSVERSSKSRPSEAKNNP SRNDAINVQGSVTANNSL FAGLRGLFHRLYSKDCWSKA
                               *      * *      *      *

Scer NLSETLFYDDSTERWVRMGEIMHY-
Spar DLSETLFYDNSTEKWKMGELVHQ-
Smik DLSKILFYDDFTEKWKMGELVHH-
Sbay DLSETLFYDDL TNRWVKMGDLVQYH
      * * * * * * * * * *

```

Figure 6.4. Multiple alignment of YBR184W shows only three conserved protein domains.

Whereas typical genes in *S. cerevisiae* show a Ka/Ks ratio of  $0.11 \pm 0.02$ , *YBR184W* has a ratio of 0.689. This ratio ranks as the third highest observed among all yeast genes (If three small domains with high conservation are excluded, the ratio rises to 0.774). The two genes with higher Ka/Ks ratio are *YAR068W*, a putative membrane protein, and *YER121W*, whose expression changes under stress.

The protein encoded by *YBR184W* has not been extensively studied, but expression studies show that the gene is induced during sporulation<sup>77</sup> and sequence analysis shows that it is similar to the gene *YSWI* that encodes a spore-specific protein. This is consistent with the observation that many of the best studied examples of positive selection in other organisms are genes related to gamete function. The change might promote speciation by imposing constraints on mating partner selection.

The vast majority of nucleotide changes in protein coding regions are silent or affect individual amino acids. However, a small number of events suggest additional mechanisms of rapid protein change. These events include closely spaced compensatory indels that affect the translation of small contiguous amino acid stretches. They also include the loss and gain of stop codons (by a nucleotide substitution or a frame-shifting indel) that may result in the rapid change of protein segments or the translation of previously non-coding regions<sup>78</sup>. Such events are observed more frequently near telomeric regions and may affect silenced genes or recently inactivated pseudogenes.

Additionally, we found a small number of differences in the length of orthologous proteins. These typically involve changes in the copy number of tri-nucleotide repeats, such as (CAA)<sub>n</sub> that encodes hydrophobic stretches often involved in protein-protein interactions. The most drastic example is seen for the TFP1 gene, which encodes a vacuolar ATPase. The *S. cerevisiae* gene contains an insertion of 1400 bp that is absent in the three related species. The insertion corresponds to the recent horizontal transfer of a known post-translationally self-splicing intein, VMA1<sup>79</sup>.

## **6.7. Conclusion.**

When comparing genomes, similarities and differences alike can reveal biological meaning. In comparing closely related species, the precise ways in which genomes change can reveal important biological insights. From the large-scale chromosomal

changes, to the substitutions of individual nucleotides, we find specific rules and constraints in the ways genomes evolve. Precise signals seem to govern how genomes are read, but also how they change. Evolutionary fitness may come from the combination of a fit genome that outperforms competition in the present, but also a modular genome that enables rapid evolution in times of extreme environmental pressure. The ability to rapidly carry out advantageous changes may be an inherent requirement in creating complexity via modularity. Evolutionary traits may be selected by reversible changes that allowed survival in the past, and will allow survival in the future. Each of the similarities and differences observed merits further experimental study. Understanding how genomes are written, and how they change, will be central to our understanding of the ever-changing book of life.

## CONCLUSION

### C.1. Summary

In this thesis, we explored the ability to extract a wide range of biological information from genome comparison among related organisms. Our results show that comparative analysis with closely related species can be invaluable in annotating a genome. It reveals the way different regions change and the constraints they face, providing clues as to their use. Even in a genome as compact as that of *S.cerevisiae*, where genes are easily detectable and rarely spliced, much remains to be learned about the gene content. We found that a large number of the annotated ORFs are dubious, adjusted the boundaries of hundreds of genes, and discovered more than 50 novel ORFs and 40 novel introns. Moreover, our comparisons have enabled a glimpse into the dynamic nature of gene regulation and co-regulated genes by discovering most known regulatory motifs as well as a number of novel motifs. The signals for these discoveries are present within the primary sequence of *S.cerevisiae*, but represent only a small fraction of the genome. Under the lens of evolutionary conservation, these signals stand out from the non-conserved noise. Hence, in studying any one genome, comparative analysis of closely related species can provide the basis for a global understanding of a wide range of functional elements.

Our results demonstrate the central role of computational tools in modern biology. The analyses presented in this thesis have revealed biological findings that can not be discovered by traditional genetic methods, regardless of the time or effort spent. Isolated deletion of every single yeast gene has been carried out without resolving the debate on the number of functional genes. Promoter analysis of any single gene could not reveal the subtle regulatory signals that become apparent at the genome-wide level. The approach presented is general, and has the advantage that one can increase its power by increasing the number of species studied. As sequencing costs lower and sequencing capacity increases, obtaining additional genomes becomes only a question of time. The comparison of multiple related species may present a new paradigm for understanding the genome of any single species. In particular, our methods are currently being applied to a kingdom-wide exploration of fungal genomes, and the comparative analysis of the human genome with that of the mouse and other mammals.



## C.2. Extracting signal from noise.

For *S. cerevisiae*, our results show that comparative genome analysis of a handful of related species has substantial power to separate signal from noise to identify genes, define gene structure, highlight rapid and slow evolutionary change, recognize regulatory elements and reveal combinatorial control of gene regulation. The power is comparable or superior to experimental analysis, in terms of sensitivity and precision.

In principle, the approach could be applied to any organism by selecting a suitable set of related species. The optimal choice of species depends on multiple considerations, largely related to the evolutionary tree connecting the species. These include the following:

(1) The branch length  $t$  between species should be short enough to permit orthologous sequence to be readily aligned. The yeasts studied here differ by  $t = 0.23$ - $0.55$  substitutions per site and are readily aligned. The strong conservation of synteny (covering more than 90% of *S. cerevisiae* chromosomes belong in synteny blocks) allowed the unambiguous correspondence of the vast majority of genes.

(2) The total branch length of the tree should be large enough that non-functional sites will have undergone substantially more drift than functional sites, thereby providing an adequate degree of signal-to-noise enrichment (SNE). For this analysis, the multiple species studied provide a total branch length of 0.83 and a probability of nucleotide identity across all four species in non-coding regions of 49%. The SNE is thus  $\sim 2$ -fold ( $=1/0.49$ ) for highly constrained nucleotides and correspondingly higher for composite features involving many nucleotides.

(3) The species should represent as narrow a group as possible, subject to the considerations above. Because the comparative analysis above seeks to identify genomic elements common to the species, it can explain only aspects of biology shared across the taxon. In the present case, the analysis identifies elements shared across *Saccharomyces sensu stricto*, a closely related set of species such that the vast majority of genes and regulatory elements are shared.

With these considerations in mind, the question remains as to what is the “right” number of species for comparative analysis. Similarly, one can ask, given a set of

previously sequenced species, what is the optimal choice for the next species to sequence. The answer of course depends on the goal at hand. In discovering genes, the number of species required depends on the length of the genes sought. In discovering motifs, the number of species depends on the motif length, its allowed degeneracy, and the total number of conserved instances. And in each case, the evolutionary distance of the species compared, but also the topology of the phylogenetic tree, will determine our ability to extract signal from noise. We found that genome-wide methods could increase the power of comparative analysis that is based on a handful of species. The answer in the general case merits a much more detailed analysis.

### **C.3. Analysis of mammalian genomes**

What are the implications for the understanding of the human genome?

The present study provides a good model for evolutionary distances (substitutions per site in intergenic regions) relevant to the study of the human. The sequence divergence between *S. cerevisiae* and the most distant relative *S. bayanus* (11% indels and 62% nucleotide identity in aligned positions) is similar to that between human and mouse (12% indels and 66% nucleotide identity in aligned positions).

An important difference between yeast and human is the inherent signal-to-noise ratio (SNR) in the genome. Yeast has a high SNR, with protein-coding regions comprising ~70% of the genome coding for protein or RNA genes and regulatory elements comprising perhaps ~15% of the intergenic regions. The human has a much lower SNR, with the corresponding figures being perhaps ~2% and ~3%<sup>19</sup>. A lower SNR must be offset by a higher SNE. Some enrichment can also be obtained by filtering out the repeat sequences that comprise half of the human genome. Greater enrichment can be accomplished by increasing the number of species studied, taking advantage both of nucleotide level divergence and frequently occurring genomic deletion<sup>19</sup>.

Such considerations indicate that it should be possible to use comparative analysis, such as explored here for yeast, to directly identify many functional elements in the human genome common to mammals. More generally, comparative analysis offers a powerful and precise initial tool for interpreting genomes.

#### **C.4. The road ahead**

In this thesis, we explored the ability of computational comparative genomics to extract biological signals that govern genes, regulation, and evolution. The nature of these signals however had been previously established experimentally. Knowing that genes were translated into amino acids every three nucleotides was central in our test of reading frame conservation. Knowing that regulatory motifs appear in multiple intergenic regions was crucial to our genome-wide discovery methods. Knowing the kinds of functional sequences to look for allowed us to examine the ways that they change. In each case, our methods relied on well-posed questions based on currently established biological knowledge.

In the future however, it will be important to formulate new hypotheses from genomic data. We cannot begin to imagine the types of information encoded in the human genome. The basis for intelligence, psychology, immunity, development, emotions are all encoded within our cells. New biological paradigms will be needed to explore novel aspects of biology, and their very discovery will reside in genome-wide studies. Development of new technologies, new statistical methods, new computational tools will be needed. An explosion of biological data, but also an explosion in novel experimental techniques has already started. And the only way to proceed is a constant marriage between biology and computer science.

## REFERENCES

1. Kowalczyk, M., Mackiewicz, P., Gierlik, A., Dudek, M. R. & Cebrat, S. Total number of coding open reading frames in the yeast genome. *Yeast* **15**, 1031-4 (1999).
2. Harrison, P. M., Kumar, A., Lang, N., Snyder, M. & Gerstein, M. A question of size: the eukaryotic proteome and the problems in defining it. *Nucleic Acids Res* **30**, 1083-90 (2002).
3. Velculescu, V. E. et al. Characterization of the yeast transcriptome. *Cell* **88**, 243-51 (1997).
4. Blandin, G. et al. Genomic exploration of the hemiascomycetous yeasts: 4. The genome of *Saccharomyces cerevisiae* revisited. *FEBS Lett* **487**, 31-6 (2000).
5. Wood, V., Rutherford, K. M., Ivens, A., Rajandream, M.-A. & Barrell, B. A Re-annotation of the *Saccharomyces cerevisiae* Genome. *Comparative and Functional Genomics* **2**, 143-154 (2001).
6. Toda, T. et al. Deletion analysis of the enolase gene (*enoA*) promoter from the filamentous fungus *Aspegillus oryzae*. *Curr Genet* **40**, 260-7 (2001).
7. Bailey, T. L. & Elkan, C. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proc Int Conf Intell Syst Mol Biol* **2**, 28-36 (1994).
8. Tavazoie, S., Hughes, J. D., Campbell, M. J., Cho, R. J. & Church, G. M. Systematic determination of genetic network architecture. *Nat Genet* **22**, 281-5 (1999).
9. Stormo, G. D. DNA binding sites: representation and discovery. *Bioinformatics* **16**, 16-23 (2000).
10. McGuire, A. M., Hughes, J. D. & Church, G. M. Conservation of DNA regulatory motifs and discovery of new motifs in microbial genomes. *Genome Res* **10**, 744-57 (2000).
11. Loots, G. G. et al. Identification of a coordinate regulator of interleukins 4, 13, and 5 by cross-species sequence comparisons. *Science* **288**, 136-40 (2000).
12. Pennacchio, L. A. & Rubin, E. M. Genomic strategies to identify mammalian regulatory sequences. *Nat Rev Genet* **2**, 100-9 (2001).
13. Oeltjen, J. C. et al. Large-scale comparative sequence analysis of the human and murine Bruton's tyrosine kinase loci reveals conserved regulatory domains. *Genome Res* **7**, 315-29 (1997).
14. Cliften, P. F. et al. Surveying *Saccharomyces* genomes to identify functional elements by comparative DNA sequence analysis. *Genome Res* **11**, 1175-86 (2001).
15. Alm, R. A. et al. Genomic-sequence comparison of two unrelated isolates of the human gastric pathogen *Helicobacter pylori*. *Nature* **397**, 176-80 (1999).

16. Carlton, J. M. et al. Genome sequence and comparative analysis of the model rodent malaria parasite *Plasmodium yoelii yoelii*. *Nature* **419**, 512-9 (2002).
17. Perrin, A. et al. Comparative genomics identifies the genetic islands that distinguish *Neisseria meningitidis*, the agent of cerebrospinal meningitis, from other *Neisseria* species. *Infect Immun* **70**, 7063-72 (2002).
18. McClelland, M. et al. Comparison of the *Escherichia coli* K-12 genome with sampled genomes of a *Klebsiella pneumoniae* and three *Salmonella enterica* serovars, Typhimurium, Typhi and Paratyphi. *Nucleic Acids Res* **28**, 4974-86 (2000).
19. Intl\_Mouse\_Genome\_Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature* **420**, 520-62 (2002).
20. Galabru, J., Rey-Cuille, M. A. & Hovanessian, A. G. Nucleotide sequence of the HIV-2 EHO genome, a divergent HIV-2 isolate. *AIDS Res Hum Retroviruses* **11**, 873-4 (1995).
21. Read, T. D. et al. The genome sequence of *Bacillus anthracis* Ames and comparison to closely related bacteria. *Nature* **423**, 81-6 (2003).
22. Genome\_Sciences\_Centre. The complete genome of the SARS associated Coronavirus. *Unpublished* (2003).
23. Goffeau, A. et al. Life with 6000 genes. *Science* **274**, 546, 563-7 (1996).
24. Galagan, J. E. et al. The genome sequence of the filamentous fungus *Neurospora crassa*. *Nature* **422**, 859-68 (2003).
25. The\_C.\_elegans\_Sequencing\_Consortium. Genome sequence of the nematode *C. elegans*: a platform for investigating biology. *Science* **282**, 2012-8 (1998).
26. Adams, M. D. et al. The genome sequence of *Drosophila melanogaster*. *Science* **287**, 2185-95 (2000).
27. Intl\_Human\_Genome\_Sequencing\_Consortium. in *Nature* 860-921 (2001).
28. Arabidopsis\_Genome\_Initiative. Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana*. *Nature* **408**, 796-815 (2000).
29. Kellis, M., Patterson, N., Endrizzi, M., Birren, B. & Lander, E. S. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature* **423**, 241-54 (2003).
30. Fitch, W. M. Uses for evolutionary trees. *Philos Trans R Soc Lond B Biol Sci* **349**, 93-102 (1995).
31. Fitch, W. M. Distinguishing homologous from analogous proteins. *Syst Zool* **19**, 99-113 (1970).
32. Tatusov, R. L., Koonin, E. V. & Lipman, D. J. A genomic perspective on protein families. *Science* **278**, 631-7 (1997).
33. Tatusov, R. L. et al. The COG database: new developments in phylogenetic classification of proteins from complete genomes. *Nucleic Acids Res* **29**, 22-8 (2001).

34. Keogh, R. S., Seoighe, C. & Wolfe, K. H. Evolution of gene order and chromosome number in *Saccharomyces*, *Kluyveromyces* and related fungi. *Yeast* **14**, 443-57 (1998).
35. Batzoglou, S. et al. ARACHNE: a whole-genome shotgun assembler. *Genome Res* **12**, 177-89 (2002).
36. Jaffe, D. B. et al. Whole-genome sequence assembly for Mammalian genomes: arachne 2. *Genome Res* **13**, 91-6 (2003).
37. Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *J Mol Biol* **215**, 403-10 (1990).
38. Thompson, J. D., Higgins, D. G. & Gibson, T. J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res* **22**, 4673-80 (1994).
39. Dujon, B. et al. Complete DNA sequence of yeast chromosome XI. *Nature* **369**, 371-8 (1994).
40. Sharp, P. M. & Li, W. H. The codon Adaptation Index--a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Res* **15**, 1281-95 (1987).
41. Clark, T. A., Sugnet, C. W. & Ares, M., Jr. Genomewide analysis of mRNA processing in yeast using splicing-specific microarrays. *Science* **296**, 907-10 (2002).
42. Batzoglou, S., Pachter, L., Mesirov, J. P., Berger, B. & Lander, E. S. Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Res* **10**, 950-8 (2000).
43. Hampson, S., Kibler, D. & Baldi, P. Distribution patterns of over-represented k-mers in non-coding yeast DNA. *Bioinformatics* **18**, 513-28 (2002).
44. Blanchette, M. & Tompa, M. Discovery of regulatory elements by a computational method for phylogenetic footprinting. *Genome Res* **12**, 739-48 (2002).
45. McCue, L. et al. Phylogenetic footprinting of transcription factor binding sites in proteobacterial genomes. *Nucleic Acids Res* **29**, 774-82 (2001).
46. Gelfand, M. S., Koonin, E. V. & Mironov, A. A. Prediction of transcription regulatory sites in Archaea by a comparative genomic approach. *Nucleic Acids Res* **28**, 695-705 (2000).
47. Lawrence, C. E. et al. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science* **262**, 208-14 (1993).
48. Grundy, W. N., Bailey, T. L., Elkan, C. P. & Baker, M. E. Meta-MEME: motif-based hidden Markov models of protein families. *Comput Appl Biosci* **13**, 397-406 (1997).

49. Hughes, J. D., Estep, P. W., Tavazoie, S. & Church, G. M. Computational identification of cis-regulatory elements associated with groups of functionally related genes in *Saccharomyces cerevisiae*. *J Mol Biol* **296**, 1205-14 (2000).
50. Roth, F. P., Hughes, J. D., Estep, P. W. & Church, G. M. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nat Biotechnol* **16**, 939-45 (1998).
51. Liu, X., Brutlag, D. L. & Liu, J. S. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pac Symp Biocomput*, 127-38 (2001).
52. Jiao, K. et al. Phylogenetic footprinting reveals multiple regulatory elements involved in control of the meiotic recombination gene, REC102. *Yeast* **19**, 99-114 (2002).
53. Tompa, M. Identifying functional elements by comparative DNA sequence analysis. *Genome Res* **11**, 1143-4 (2001).
54. Blanchette, M., Schwikowski, B. & Tompa, M. Algorithms for phylogenetic footprinting. *J Comput Biol* **9**, 211-23 (2002).
55. Keegan, L., Gill, G. & Ptashne, M. Separation of DNA binding from the transcription-activating function of a eukaryotic regulatory protein. *Science* **231**, 699-704 (1986).
56. Zhu, J. & Zhang, M. Q. SCPD: a promoter database of the yeast *Saccharomyces cerevisiae*. *Bioinformatics* **15**, 607-11 (1999).
57. Zhang, M. Q. Promoter analysis of co-regulated genes in the yeast genome. *Comput Chem* **23**, 233-50 (1999).
58. Mewes, H. W. et al. MIPS: a database for genomes and protein sequences. *Nucleic Acids Res* **27**, 44-8 (1999).
59. Gasch, A. P. & Eisen, M. B. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. *Genome Biol* **3**, RESEARCH0059 (2002).
60. Simon, I. et al. Serial regulation of transcriptional regulators in the yeast cell cycle. *Cell* **106**, 697-708 (2001).
61. Lee, T. I. et al. Transcriptional regulatory networks in *Saccharomyces cerevisiae*. *Science* **298**, 799-804 (2002).
62. Mewes, H. W. et al. MIPS: a database for genomes and protein sequences. *Nucleic Acids Res* **30**, 31-4 (2002).
63. Gavin, A. C. et al. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature* **415**, 141-7 (2002).
64. Dwight, S. S. et al. *Saccharomyces* Genome Database (SGD) provides secondary gene annotation using the Gene Ontology (GO). *Nucleic Acids Res* **30**, 69-72 (2002).

65. Mosley, A. L., Lakshmanan, J., Aryal, B. K. & Ozcan, S. Glucose-mediated phosphorylation converts the transcription factor Rgt1 from a repressor to an activator. *J Biol Chem* (2003).
66. Lindgren, A. et al. The pachytene checkpoint in *Saccharomyces cerevisiae* requires the Sum1 transcriptional repressor. *Embo J* **19**, 6489-97 (2000).
67. Jacobs Anderson, J. S. & Parker, R. Computational identification of cis-acting elements affecting post-transcriptional control of gene expression in *Saccharomyces cerevisiae*. *Nucleic Acids Res* **28**, 1604-17 (2000).
68. Zeitlinger, J. et al. Program-specific distribution of a transcription factor dependent on partner transcription factor and MAPK signaling. *Cell* **113**, 395-404 (2003).
69. Gardner, M. J. et al. Genome sequence of the human malaria parasite *Plasmodium falciparum*. *Nature* **419**, 498-511 (2002).
70. Wolfe, K. H. & Shields, D. C. Molecular evidence for an ancient duplication of the entire yeast genome. *Nature* **387**, 708-13 (1997).
71. Fischer, G., Neuveglise, C., Durrens, P., Gaillardin, C. & Dujon, B. Evolution of gene order in the genomes of two related yeast species. *Genome Res* **11**, 2009-19 (2001).
72. Fischer, G., James, S. A., Roberts, I. N., Oliver, S. G. & Louis, E. J. Chromosomal evolution in *Saccharomyces*. *Nature* **405**, 451-4 (2000).
73. Dunham, M. J. et al. Characteristic genome rearrangements in experimental evolution of *Saccharomyces cerevisiae*. *Proc Natl Acad Sci U S A* **99**, 16144-9 (2002).
74. Bon, E. et al. Genomic exploration of the hemiascomycetous yeasts: 5. *Saccharomyces bayanus* var. *uvarum*. *FEBS Lett* **487**, 37-41 (2000).
75. Haber, J. E. Mating-type gene switching in *Saccharomyces cerevisiae*. *Annu Rev Genet* **32**, 561-99 (1998).
76. Hurst, L. D. The Ka/Ks ratio: diagnosing the form of sequence evolution. *Trends Genet* **18**, 486 (2002).
77. Chu, S. et al. The transcriptional program of sporulation in budding yeast. *Science* **282**, 699-705 (1998).
78. True, H. L. & Lindquist, S. L. A yeast prion provides a mechanism for genetic variation and phenotypic diversity. *Nature* **407**, 477-83 (2000).
79. Koufopanou, V., Goddard, M. R. & Burt, A. Adaptation for horizontal transfer in a homing endonuclease. *Mol Biol Evol* **19**, 239-46 (2002).



## APPENDIX

**Counting combinations:** The number of ways to choose  $k$  items without replacement from a total of  $n$  is given by ( $n$  choose  $k$ ) :

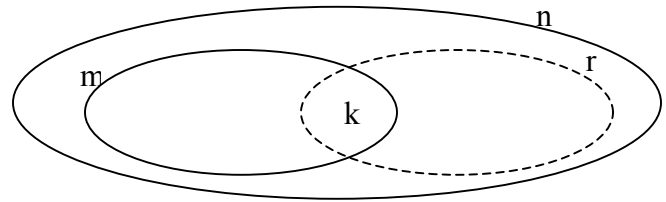
$$\binom{n}{k} = \frac{n!}{(n-k)!k!} = \frac{n(n-1)\dots(n-k+1)}{k!}$$

**Binomial distribution:** The probability of obtaining  $k$  successes out of  $n$  trials given a probability  $p$  of success for any one trial is given by:

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

**Hypergeometric distribution:** When choosing a random subset of size  $r$  from  $n$  items of which  $m$  belong in a particular category, the probability that  $k$  of the selected items belong to that category is given by:

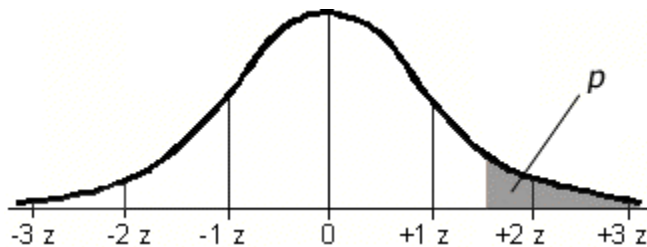
$$p(X = k) = \frac{\binom{m}{k} \binom{n-m}{r-k}}{\binom{n}{r}}$$

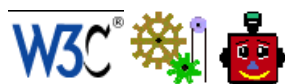


**Standard normal distribution (or Gaussian distribution):** The sum of a large number of independent variables follows a normal distribution of density function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

**Computing z-scores:** Any probability  $p$  can be represented as the standard deviations away from the mean of a standard normal distribution corresponding to tail area  $p$ .





# Constraint Model for libwww Webbot

[Manolis Kamvysselis](#) - [Henryk Frystyk Nielsen](#)  
[World Wide Web Consortium](#) - Summer 96

Jump to:

- [Design](#) issues and purpose
- [Rules](#) description and issuing of commands
- [URLs](#) and Web Elements
- [Function](#) description for extending the robot
- [Extensions](#) to the robot, and what's next

## I. Design issues and Purpose of the Robot

The W3C robot's purpose in crawling the web is not to accumulate data from web sites, to generate a database, or an index, or to update documents, or verify links, even though all of the functionality described above can easily be added, without changing the basic skeleton of the robot as it is today.

The primary purpose of the robot is to understand the relations between web pages, and not the content of the pages themselves. How many paths can lead to a given page, what tree structures can be generated by following links under a certain constraint, how many degrees of separation lie between any two pages, what is the shortest path to reach a site, and how can intelligent navigation be automated.

The robot will also be used as a test tool interacting with the w3c-libwww-99. The robot code is written in Tcl, the graphic interface to the user is written in Tk, and the library code used to provide all the network functionality is written in C. All functionality is therefore portable in almost any system with network access.

The most elementary web robot can fetch a web document, extract the links from that document, follow the urls, fetch the documents, and restart for every document. This leads to an exponentially growing number of documents to follow every time you reach a new depth. Therefore, for a robot to be of any use, we need a way of specifying a **Constraint Model** that will allow the robot to choose which documents to follow and which not to.

The Constraint Model is basically a set of rules, that the user specifies, or that the program itself can formulate, to follow a crawling model specified by the user at the beginning, or during the crawling. The next section describes the constituent blocks of the rules (or Rule Atoms) and the ways to combine those rules.

## II. Rule Description

A rule can be simple or complex. Complex rules can be expressed as logical combinations of simpler rules, called Rule Atoms.

### A. Rule Atoms

#### Definition

A Rule Atom is the simplest form of a rule, and the building block of every rule. Anything simpler can not be called a rule, and anything more complicated cannot be called a rule atom.

#### Examples

```
+url:*.mit.edu/*
  keep documents whose url matches *.mit.edu/*
  matches: http://web.mit.edu/manoli/www/ and http://manoland.mit.edu/ but not http://web.mit.edu.ca/
-body:*badword*
  reject documents whose body contains badword
```

```
+head:*meta*
  keep documents whose head contains the meta keyword
-date:*december*
  reject documents whose date contains the word december
```

## Understanding the Rule Atoms

As you can see from those examples, each **rule atom** has three elements:

- a. The **category** of the rule can be one of the following four, in order of priority:
  1. **Imperative rules (\*)**: those documents will be followed first and always  
**Direction rules (+)**: Keep: the documents will be followed next if they don't match -  
**Rejection rules (-)**: Reject: document will not be followed, unless it matches an imperative rule  
**LastResort rules (?)**: Default: if the robot has no more documents matching \* or +, it follows ?
- b. The **field** of the document to search
  1. Sample fields are: url, head, body, date, etc...  
 The program calls `Url_Geturl`, `Url_Getbody`, etc, depending on the field specification  
 To add a new field, all you need to do is provide a function
- c. The **pattern** to match
  1. For the moment, it is a glob-type pattern: \* matches any combination of characters, ? matches one character. Later on, functionality might be added, to handle regular expression matching.  
**Please Note:** If you want to match the pattern in the middle of a string, you have to add an initial and final \* Here are some examples, that you should take a look at:
    - \*.mit.edu does not match `http://web.mit.edu/` since the final \* is not present
    - \*.mit.edu/\* matches `http://web.mit.edu/people/`, `ftp://www.mit.edu/pub/`
    - \*.edu\* matches `http://www.sfu.edu.co/`, `ftp://www.mit.edu.ca/pub/bin/`

## B. Combining Rules to form Rules

### Definition of a Rule

The definition is recursive: a rule can be either a rule atom, or a logical combination of rules.

### Undersating Rule Combination

Here are the forms a rule can take and the result of applying the rule to a list of documents

Applying ... to a list of documents	Returns the list of documents that...
RuleAtom	match an imperative rule, or that match a directional rule and don't match a rejection rule
{OR rule1 rule2 ...}	either returned by applying rule1, or returned by applying rule2, or ..
{AND rule1 rule2 ...}	urls returned by rule1, and by rule2, and by...
{NOT rule1 rule2}	urls returned by rule1 and not by rule2.

### Example

With those simple rules, we can express very precise constraints

```
{NOT {AND +url:*.mit.edu/* -url:*.mit.edu/personnal/*
      {OR -body:*badword* +head:*goodcontext*}}
+date:*october*}
```

## C. Applying Rules

### The notion of truth

Applying a rule such as `+url:*.mit.edu/*` to a list of elements

more...

Please note that the logical NOT operator is not defined in its common interpretation of a unary operator, negating something that is false, to give something true. This is because testing a rule against a set of web elements, does not return true or false, but the list notion of true and false, which is sublists. Therefore, negating a list list2 cannot return the contrary of list2, unless we know what represents the "truth" which in this case is list1. Therefore, one must understand that we can only negate a list relative to a larger list. {NOT 1} is always 0 and {NOT 0} is always 1, but the result of evaluating a rule against a list of documents (that we will assimilate to URLs in this paragraph) is

not a boolean, but a sublist. Therefore, if `http://web.mit.edu/` is the result of applying rule2, the `{NOT rule2}` will be `{NOT http://web.mit.edu/}`, which would represent the entire web, except `http://web.mit.edu/`. One must therefore provide the part of the web against which `http://web.mit.edu/` will be negated, and that in form of a list.

### Applying a rule to an Element list

- a. Reminder
  1. As you saw earlier, a rule atom has the form `+url:*.mit.edu/*`, and is formed of the category specification (`*`, `+`, `-`, `?`), the field to check (url, head, body, date, etc) and the pattern to match (`*.mit.edu/*` in this example).
- b. Applying a Rule Atom to a Web Element
  1. Depending on the field specification of the rule, a corresponding `Get$field` function is called on the Web Element (where `$field` can be any of url, head, body, date, etc...), and the pattern is matched against that result. To be able to match against more fields, all one has to do is provide the code for getting the corresponding field from the Web Element, and the rule will be used with no difference. For example, to be able to use a rule `+hat:top` or `+hat-color:red`, all one has to do is provide the functions `URL_Gethat` and `URL_Gethat-color`. The name can be anything, but cannot contain a column (`:`), since it is used to separate the field from the pattern in a rule.
- c. Applying a Rule Atom to a List of Web Elements
  1. When the Web Atom is applied to a List of Elements, two local lists are created, named `matched`, and `unmatched`, and each Element checked goes to one list or the other. After every element in the list has been checked, the result (the return value) of applying the Rule to the Element List is the list of matched elements in the case of an imperative (`*`) or directional (`+`) rule atom, or the list of unmatched elements, in the case of a negational (`-`) rule.
- d. Applying Rule Combinations to Element Lists
  1. When applying `{AND rule1 rule2 {OR rule3 rule4}}` to a list of elements, rule1, rule2, rule3, and rule4 are applied separately to the list of elements, yielding list1, list2, list3, list4. Then is evaluated the expression `[List_and list1 list2 [List_or list3 list4]]`, which returns the desired list value, the result of applying the rule combination to the element list.
- e. Accepting or rejecting an element

## D. Sets of Rules

### Global Rules and Local Rules

- a. Design issues and purpose
  - The Constraint Model of the Robot was built on the idea that the links a user will choose to follow, do not only depend on the purpose of his navigation and his choices of sites, but also, on the history of his navigation, the documents accessed previous to the current page. Therefore, a constraint model would be more efficient by having not only a set of static, global rules, that are set by the user when the navigation begins, but also a set of local rules for each page, that are set automatically, as the navigation progresses, and that are specific to a page, or a family of pages.
- b. Global Rules and Constraints
  - Global rules are set by the user before the navigation begins. They represent the rules that the program has to follow throughout his navigation, and cannot be overwritten by local rules. For example, if a user doesn't want to access personal pages, a global rule would be `-url:*/Personnal/*`, or if he doesn't want to access only educational pages, a rule might be `+url:*.edu*`
- c. Local Rules and Navigation method
  - The local rules are set every time a new element is created. They are set based on a global variable, `nav`, that the user sets at the beginning of the navigation, and that he can change at any time. This variable can now have four values: `dumb`, `deep`, `broad`, `strategic`.

## III. Representing Documents and Urls

### A. A Web Element

#### Definition

A Web Element is the sum of all the information gathered on a certain web document reached following a certain path, with a known set of rules.

#### Representation of a Web Element

A Web Element consists of:

**Family**

The documents reached from parent 0302 will have family indexes 030201, 030202, 030203. The family index is unique, and fully identifies a given Element.

**Url**

The same document can be reached from two different paths. The same url therefore can be shared by 030202 and 0401.

**Priority**

It shows how important a certain element is for the navigation of the robot. Here's a table of values:

Value	Category	Corresponding rule	Description
0	Untested	none	Element has not been tested yet
1	First	imperative	Will be followed first
2	Next	directional	Followed next
3	Last	lastresort	Followed last
5	Rejected	rejection	Never followed

**Rules**

The local rules of the url. Updated automatically.

**State**

Has the url been followed yet. 1 for yes, 0 for no.

**B. The Web**

As was said earlier, the main interest of the robot, is not how the web is constructed, how a site is managed, or how the web should be ordered, but the relations that exist among documents,

## IV. Program Functions

### A. Rule Functions

function	input	outputdescription
Rule_Init		Initializes rules
Rule_ShowAll		
Rule_ApplyAtom	elementlist ruleatom	list of matching elements
Rule_Apply	elementlist rule	
Rule_and		
Rule_or		

### B. Handling Urls

function	input	outputdescription
Url_Init		Initializes urls
Url_ShowAll		
Url_ApplyAtom	elementlist ruleatom	list of matching elements
Url_Apply	elementlist rule	

### C. Handling Lists

function	input	outputdescription
List_and	list1 list2 ...	list of common elements to list1 list2 ...
List_or	list1 list2 ...	list of elements either in list1 or list2 ...
List_not	list1 list2	list of elements in list1 and not in list2

## V. Extending the Robot

#### Gathering more information

What is really easy to do with the current design of the robot, is to make it gather more information from the documents that it accesses. All one has to do is change the `Url_Load` function.

#### Graphical Representation

Once elements are gathered by the robot, their family indexes describe their relations very precisely. One can then choose a representation for trees, and provide a graphical representation of the portion of the web accessed.

#### Distributing Web indexing and crawling

When a stable design of storing results is reached and proven general and extensible, then different parties can run their robot, and create tree structures inside their own sites, and store these results in an agreed-upon location, that will be searched by the robot, when accessing that site, and the results will be reused.

#### Strategic navigation

The present robot doesn't know how to handle strategic navigation, which would be trying to reach a url. A special way of calculating rules will have to be formulated.

#### More

And much much more...

---

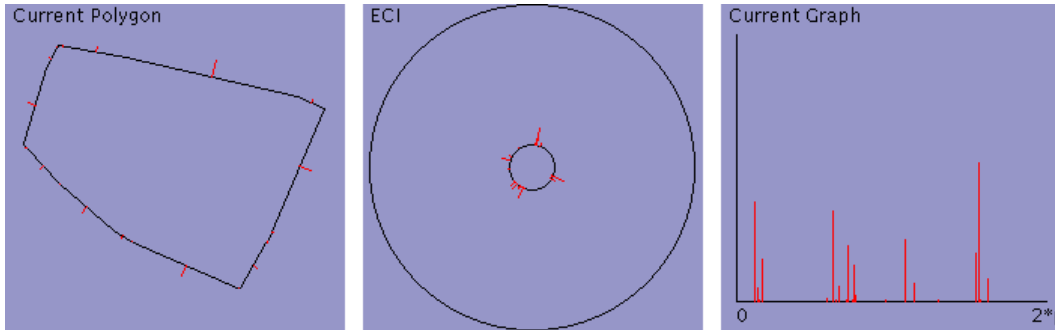
[Manolis Kamvysselis](mailto:manoli@mit.edu) - [manoli@mit.edu](mailto:manoli@mit.edu) - <http://mit.edu/manoli>  
*Summer 96*

# 2D Polygon Morphing

using the

## Extended Gaussian Image

Manolis Kamvysselis



- |                                 |                              |                                   |                            |
|---------------------------------|------------------------------|-----------------------------------|----------------------------|
| 1. <a href="#">Introduction</a> | 2. <a href="#">Algorithm</a> | 3. <a href="#">Implementation</a> | 4. <a href="#">Results</a> |
| • Abstract                      | • Overview                   | • <a href="#">Program</a>         | • Pictures                 |
| • Prior Work                    | • Moving Weights             | • Using the program               | • Types of morphing        |
| • Motivation                    | • Matching Weights           | • <a href="#">Source Code</a>     | • Future Work              |

## Introduction

### Abstract

Morphing techniques that have been developed to progressively transform one two-dimensional image to another are usually pixel based and morph a source to a target by interpolating pixel values based on constraints specified by the user.

In this work, we are using a model-based approach to propose a solution to morphing a two-dimensional polygon into another. Our model of an object will be its Extended Circular Image representation, the two-dimensional equivalent of the Extended Gaussian Image. This approach will work only for convex polygons, for which the ECI is unique.

To morph non-convex polygons, one would have to separate them into convex components and morph each separately, or use an extended version of the ECI that would apply to non-convex polygons. This morphing method can be applied to convex polyhedra using the Extended Gaussian Image, and similarly be extended to non-convex polyhedra.

### Prior work

This term, I was working on a model-based three-dimensional morphing project for Computer Graphics.

[The algorithm](#) I developed for that project was simply matching polygons in a three-dimensional world in a nearest-neighbor fashion, and [growing unmatched polygons](#) from zero-area polygons on edges of other polygons.

[The results](#) of this approach were often quite [satisfactory](#) both graphically and mathematically, the polygons indeed traveling the way I hoped they would. However, the algorithm never assumed that the source or target objects were connected, and sometimes broke the [connectivity](#) which should have been preserved.

### Motivation

Having explored the unconnected case, I decided to explore the other extremity, where we assume that all polygons are connected. Including the additional constraint that those closed objects are convex, we can use the Extended Gaussian Image representation of an object to uniquely define it in 3D.

The problem therefore becomes that of moving the weights of the EGI on the unit sphere to redistribute them such that the EGI of the source polyhedron becomes that of the target, while keeping the center of mass at the origin (so that all intermediate objects are closed themselves).

This approach allows us to get around the problem of matching two objects of different connectivity by letting the inverse EGI function create

faces of arbitrary shape, our program having specified only their size and normal direction by moving the weights on the three dimensional sphere.

The current algorithms for computing the inverse EGI being incremental, we are optimistic about this approach since the difference in connectivity and shape between two morphing steps will generally be small, allowing a faster computation of the inverse EGI.

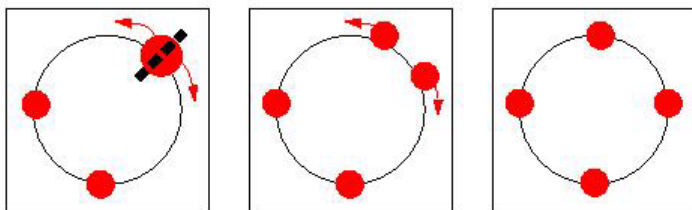
## The Algorithm

The current two-dimensional algorithm is

1. First computing the ECI of the source and of the target polygons
2. Then matching source and target normals on the ECI circle creating source-target normal pairs
3. It then linearly interpolates weights and angles between normal pairs to derive the ECI of intermediate steps
4. Finally, it reconstructs the convex polygon corresponding to the ECI obtained by interpolating the normals.

## Moving Weights

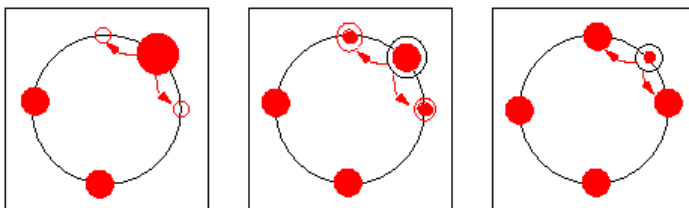
Many approaches are possible for transforming the source normals to the target normals. Each corresponds to a different way of picturing the problem to solve. All approaches must make sure that the weights have their center of mass at the center of the unit circle at every intermediate step, such that convexity is preserved.



**Case 1**  
Chop and Move

Separate weights into pieces and change only angles of pieces at every step.

One could picture the normals as physical weights on a sphere and thus want to simply interpolate angles of the normals, unable to take weight from one normal to another across the sphere. This approach will break each source weight into many chunks and it will then move those pieces of constant weight around the sphere until they recombine into the target ECI.



**Case 2**  
Shrink and Grow

Normals can exist only at specified points. Teleport weights at every step.

Alternatively, one could consider the angles of the normals as the only places on the sphere where a weight could appear, as liquid containers all connected to each other, the weight being the quantity of liquid that we would pour into each container. We'll then simply transfer weights between normals at every step.

## General Case

We'll describe the general case as a combination of those two basic movements described above. Every normal will be allowed to move on the sphere, while distributing its weight to other normals and collecting weight from them. One can integrate both movements by matching initial and final angle and weight for every normal increment, and linearly interpolating both angle and weight. One has to then prove that the center of mass of the increments is the center of the ECI circle, to guarantee that every intermediate step will be a closed polygon.

Except from how much the angles will change and how much the weights will change, one has to make more decisions, such as which normals are affected by which others. Many choices are possible, ranging from the case where every normal is affected by and affects only the normals closest on its right and left (could be only two, or all those needed to sum up the normal's weight), to the case where every normal affects and is affected by all the other normals.

Those decisions will lead to intermediate objects with different characteristics. Intermediate edges can be small, yielding smooth objects, circular at the extreme case, or they can be larger, minimizing creation and deletion of faces, and thus conserving the topological similarities between source and target as much as possible. The right balance between smooth transitions and topology preservation is often an aesthetic decision of the



user, and neither approach can be argued more correct than the other.

## Matching Weights

In our approach, each normal is compared with all other normals, and a cost  $c(i,j)$  of transforming every source to every target is calculated as follows:

$$c(i, j) = \sigma \cdot (\Delta_{i \rightarrow j}(x) + \lambda \cdot \Delta_{i \rightarrow j}(\alpha))$$

where  $\Delta(a)$  is the minimum angle difference between two normals (cycling around  $2\pi$  if necessary), and  $\Delta(w)$  is the weight difference between the normals. Both deltas are scaled to fit a  $[0..1]$  range, such that matching is independent of the coordinate system.

$$\Delta_{i \rightarrow j}(\alpha) = \frac{1}{\max \Delta \alpha} \cdot \min(|\alpha_j - \alpha_i|, |(2\pi + \alpha_i) - \alpha_j|)$$

$$\Delta_{i \rightarrow j}(x) = \frac{1}{\max \Delta \text{weight}} \cdot |\text{weight}(i) - \text{weight}(j)|$$

Moreover, we use a lambda factor to multiply one of the two differences (here, the angle), such that a change in angle costs more than an equivalent change in weight, so that we can emphasize the properties we'd like to see preserved in our morphing.

Those costs are then entered into a function that guarantees that the center of mass of the ECI is preserved throughout the morphing process, and that intermediate polygons will also be closed. I owe this function to Panayiotis (Peter) Kamvysselis (pkamvyss@mit.edu), my older brother, with whom I discussed this interesting problem of matching normals on a unit sphere for morphing, and whose ideas and opinions were very helpful. We calculate the source normal portion corresponding to the part of the  $i$ -th normal that goes to the  $j$ -th source, by scaling the  $i$ -th normal by  $w(i \rightarrow j)$  below, and the target normal portion received by the  $j$ th target from the  $i$ th source is obtained by scaling the  $j$ th target by  $w(j \leftarrow i)$  below. The function used is: >

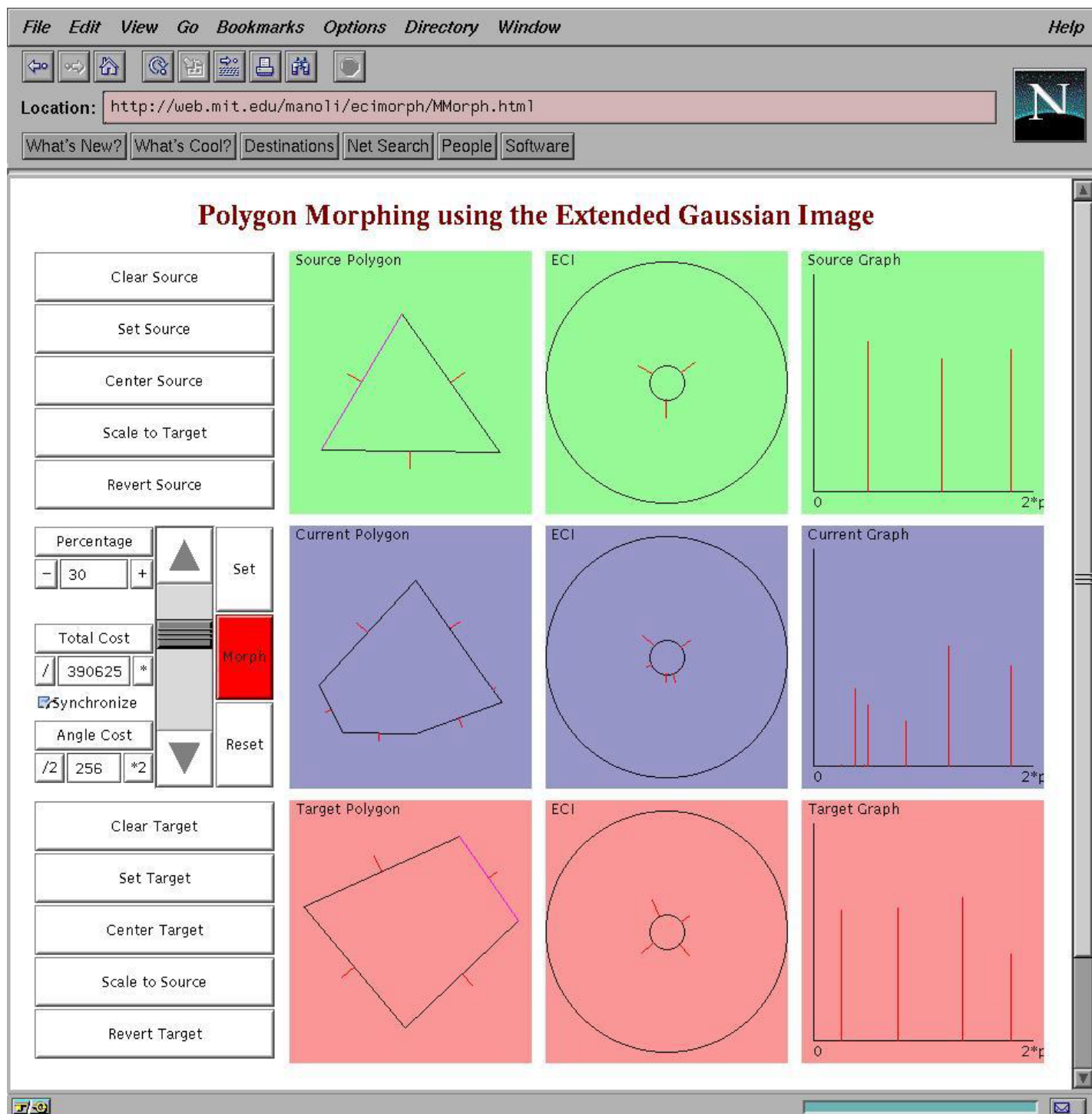
$$w_{i \rightarrow j} = \frac{e^{-c(i, j)}}{\sum_j e^{-c(i, j)}} \quad w_{j \leftarrow i} = \frac{e^{-c(i, j)}}{\sum_i e^{-c(i, j)}}$$

This function allows for each normal to be morphed into only one other normal, if the match is very good, or contribute slightly to all other normals, if no good match exists. The appearance of the intermediate objects therefore depends on the quality of the existing matches, according to the criteria specified by the lambda above, but also on the variance  $1/\sigma$  of the function, which appears as a scale factor of the total cost  $c(i,j)$ .

## Implementation

### Program

As a proof of concept, you can find a two-dimensional version of the algorithm at [MMorph.html](http://web.mit.edu/manoli/ecimorph/www/ecimorph.html).



## Using the program

### To morph two polygons

- Enter the source polygon (press "Clear Source" then click once for every vertex then "Set Source")
- Enter the target polygon in the same fashion
- Press Set (above the Morph button). This calculates the increments.
- Move the scrollbar to choose morph percentage. This simply does the interpolation.

### Parameters

- Percentage
  - Allows more precision than the scrollbar for choosing morph percentage.
- Total Cost

- Sigma in the formula below. Compensates for individual scaling.  
Angle Cost  
Lambda in the formula below. Cost of a change in angle when change in weight costs 1.
- $$c(i, j) = \sigma \cdot (\Delta_{i \rightarrow j}(x) + \lambda \cdot \Delta_{i \rightarrow j}(\alpha))$$
- Synchronize  
Automatically adjusts total cost such that:  $\sigma * (\dots + \lambda) = 100$

## Buttons

- Clear Source/Target  
Clears the temporary data of the polygon and lets you enter points
- Set Source/Target  
Reads in the temporary data drawn by the user. Calculates ECI.
- Center Source/Target  
Translates the data such that bounding box and panel share center
- Scale Source/Target  
Scales the data such that source and target perimeters are the same.
- Revert Source/Target  
Replaces temp data with stored data.

## Code

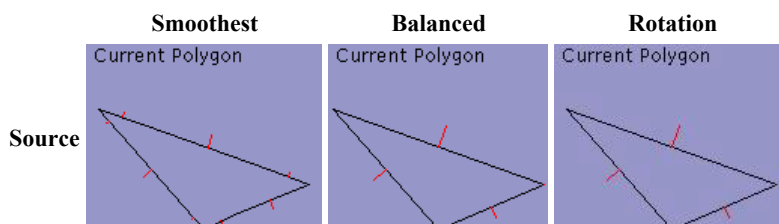
The Java byte code is available at MM\*.class, and the source is at MM\*.java.

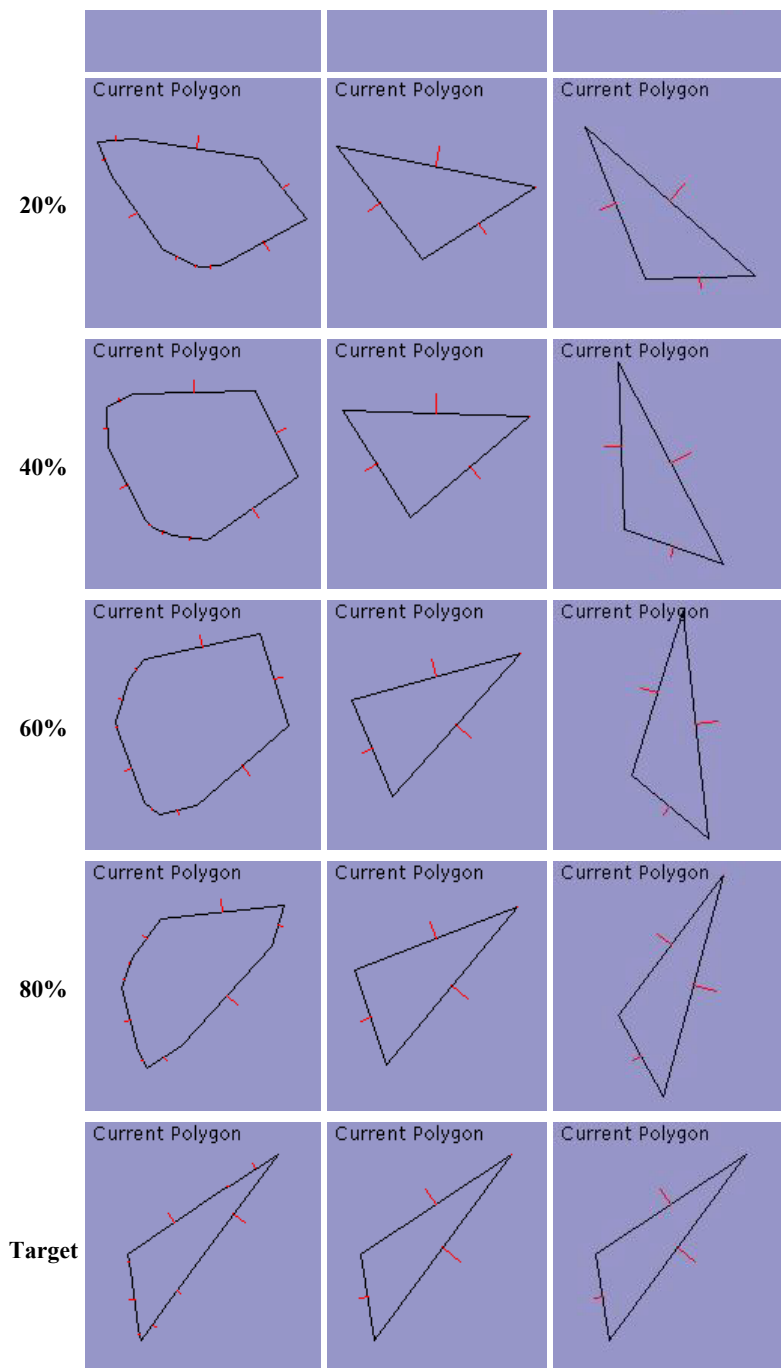
The following links will lead you to the source code of individual files of the code of the program. Feel free to use part or all of it, while giving credit to the author (Manolis Kamvyselis -- manoli@mit.edu -- http://web.mit.edu/manoli/www/).

- [MMorph.java](#) -- [MMorph.class](#)  
Construction of the GUI. Set up of objects. Basic calls to respond to buttons.
- [MMobject.java](#) -- [MMobject.class](#)  
Basic object. Contains data, polygon representation, ECI representation, EGI graph
- [MMpoly.java](#) -- [MMpoly.class](#)  
The polygon representation. Draws the points of a polygon. Accepts mouse input to set polygon.
- [MMeci.java](#) -- [MMeci.class](#)  
Draws the ECI representation on the unit circle.
- [MMgraph.java](#) -- [MMgraph.class](#)  
Lays down the angle axis of the ECI from 0 to 2pi and plots the normals.
- [MMdata.java](#) -- [MMdata.class](#)  
The data of all reps. Coordinates, normals, angles, weights.
- [MMpoint.java](#) -- [MMpoint.class](#)  
The (x,y) coordinates and methods
- [MMnormal.java](#) -- [MMnormal.class](#)  
The (angle, weight) and more of a normal vector.
- [MMmnormal.java](#) -- [MMmnormal.class](#)  
A morphable normal we construct from source and target normals. Contains initial point and increments.
- [MMmdata.java](#) -- [MMmdata.class](#)  
Morphable data. Contains morphable normals and reconstructs

## Results

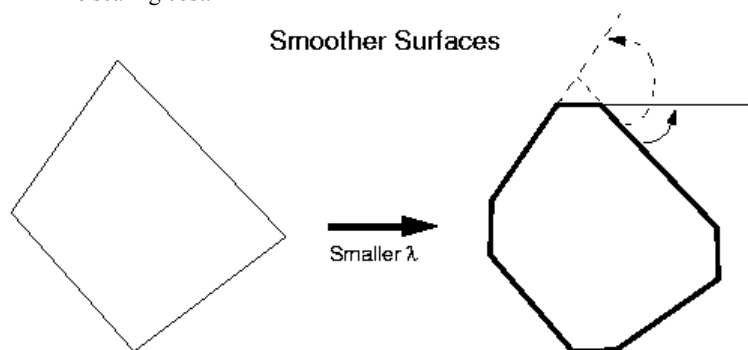
### Types of morphing





### Smooth

When  $\lambda$  (which multiplies the angle change) is small (less than 1), what really matters in our cost function is the change in weights, by the [formula](#) above. The program will therefore break the edges into components which don't have to be scaled, but simply rotated, such as to minimize scaling cost.



Since the angle of the edge components will be interpolated linearly, from their source orientation to their destination orientation, the external angles shown in the diagram above will be minimized, yielding smoother intermediate polygons, with smoother angle transitions.

## Balanced

When  $\lambda$  is larger, the intermediate polygons tend to have larger edges. The program will match source normals to target normals with nearby angles, even if the weights have to be scaled, since the cost of scaling decreases as the cost of rotating increases. The second column above shows a balance between rotation and scaling obtained with  $\lambda = 1$ .

## Rotation

The same source and target triangles presented in the two previous examples can also be transformed into one another by a simply rotation, which the program has found by increasing the angle cost a little more. This transformation, obviously minimizing weight change, apparently also minimizes rotation angle, since only three sides are rotated.

## More

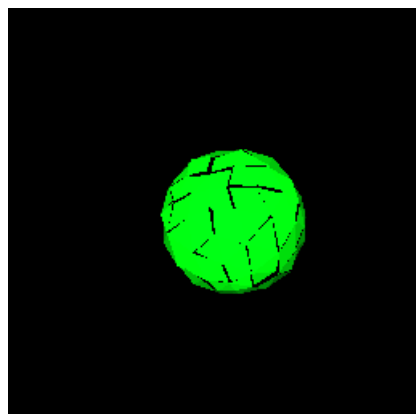
Feel free to [play around](#) with more examples.

## Future Work

The results of this project leave us optimistic about the three-dimensional case, at least for the normal matching on the unit sphere. The recovery of the polyhedron from the interpolated Gaussian Image is still generally little known and expensive, but we have interesting ideas for improving the reverse EGI operation based on the tons of information provided by known polyhedral models of neighboring EGIs.

---

Prof. Berthold K P Horn - [Manolis Kamvysselis](#) - [manoli@mit.edu](mailto:manoli@mit.edu) - December 20, 1997

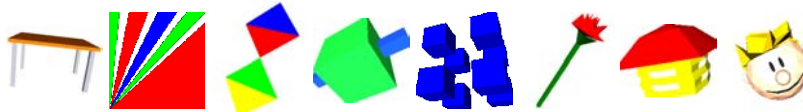


Jump to:  
[Pipeline](#)  
[Ideas](#)  
[Rendering](#)  
[Algorithm](#)  
[Future](#)  
[Examples](#)  
[Slides](#)

## 3D Morphing

[Manolis Kamnitsis](#) - [Matt D. Blum](#) - [Hooman Vassef](#) - [Krzysztof Gajos](#)  
 { [manoli](#) - [mdblum](#) - [hvassef](#) - [kgajos](#) } @mit.edu

Click on the icons below to see morphing animations



## Abstract

*This paper presents our work towards achieving a model based approach to three dimensional morphing. It describes the initial algorithms and ideas that we envisioned, the final algorithm we developed and implemented, the environment we worked in, our visualization techniques, and future work planned on the subject.*

## Introduction: Different types of morphing

Morphing is an interpolation technique used to create from two objects a series of intermediate objects that change continuously to make a smooth transition from the source to the target. Morphing has been done in two dimensions by varying the values of the pixels of one image to make a different image, or in three dimensions by varying the values of three-dimensional pixels. We're presenting here a new type of morphing, which transforms the geometry of three dimensional models, creating intermediate objects which are all clearly defined three-dimensional objects, which can be translated, rotated, scaled, zoomed-into.

### Two-dimensional morphing

Two-dimensional morphing is transforming an array of  $m$  by  $n$  pixels into another array progressively. An intermediate value between two pixels can be obtained by interpolating  $rgb$  values of the source and end pixels in more or less complicated ways. However, straight color interpolation creates many unwanted side effects such as ghosting and unnatural transitions.

A better way to accomplish two-dimensional morphing is to identify line segments on the source image with line segments on the target image so that pixel values will actually move across the image so that features will be preserved better. For example to map a face to another face, it is important that certain features such as the eyes, nose, and mouth are identified so that intermediate images actually look natural. The mouth of the source image will move to the proper place in the target image.



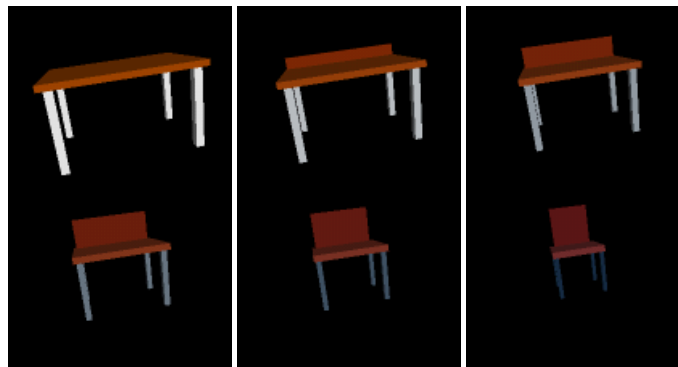
Figure 1: Two dimensional morphing of images

### Prior three-dimensional morphing

Three dimensional morphing has been done using more or less the same technique. Instead of dealing with pixels in a two-dimensional image though, **the people who did this** used pixels in a three dimensional structure. The algorithms however are still the same, the features identified being now points, edges, cubes, and other three-dimensional structures.

### Our approach

Our approach is different however, from any prior work we could find on morphing. We are transforming objects, and not interpolating pixels. We are dealing with transforming the geometry itself of an object. Our representation of a three-dimensional object is a union of the triangles in its triangulation, and our goal is an algorithm that maps triangles in the source object to triangles in the target.



**Figure 2:** Three dimensional morphing of objects

The three-dimensional morpher creates intermediate scenes, calculating the geometry of every scene that falls between the source and the target. In **Figure 2** (obtained using our program), every intermediate object between a table and a chair, is an object by itself, and it's hard to know when the object stops being a table and starts being showing smooth transitions of geometry as well as the color varying continuously.

It is easy to describe the basic algorithm in words, but such a mapping scheme of triangles is harder than one would initially think. What happens for example if the two scenes contain different numbers of triangles or the triangles are connected in different ways in the two scenes? That is, how do we handle topologically different objects?

## Outline

Our primary goal was to create a program by which we could morph two polyhedral objects into one another. for the morphing algorithms, we considered [several approaches](#) before implementing the [final algorithm](#).

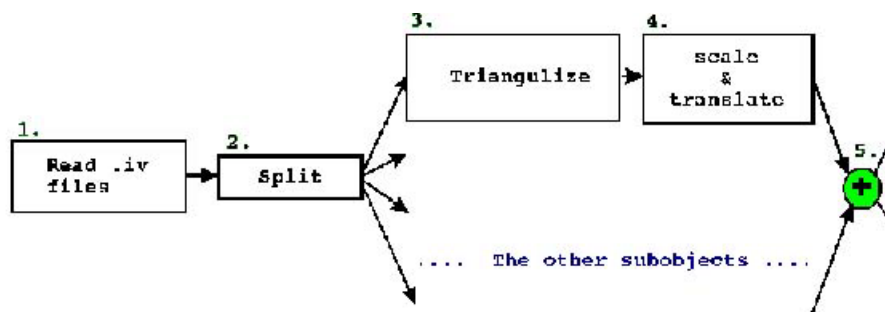
Using the SGI's as a platform, we decided to take to our advantage the Open Inventor library, including the triangulation routine used in the `ivscan` program. Our program takes as an input a number of frames  $k$  and two Inventor files, preferably written following some basic rules. It feeds them trough a [pipeline](#), and displays the  $k$  intermediary objects that it created using the [morphing algorithm](#) between the source and target objects. Also, using an [off-screen renderer](#), we can create a sequence of `*.gif` files (see **Figure 2**).

## Overview of the implementation - by Krzysztof Gajos

The structure of our program is pipelined and component oriented. That architecture was dictated by the nature of our solution to the program. Incidentally, it made the implementation process easier because fully functional components could be introduced and tested independantly provided stubs were provided to simulate other, not yet implemented, parts of the program. Also, many components that output objects of the same type as they took as an input, could be skipped completely.

It has to be mentioned at this point, that certain components of our program are adaptations of the IvScan code.

Below there is a schematic diagram representing the architecture of our program. A more detailed explanation of each component follows.





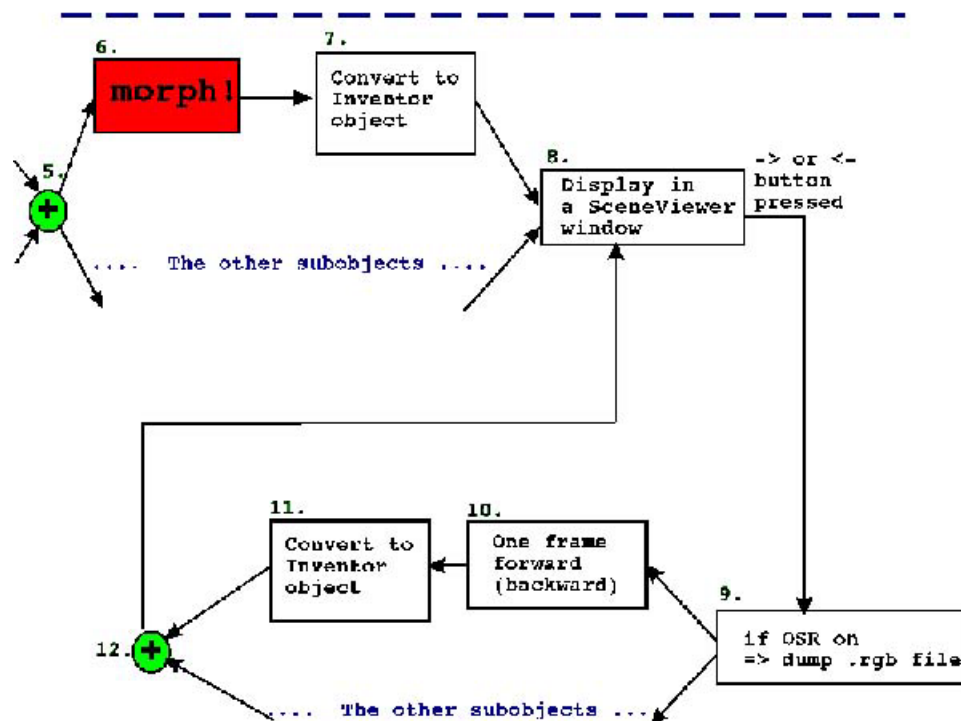


Figure 3: Schematic diagram of structure of our program.

**1. Read .iv files** Reads the source and target files in Inventor format. Builds source and target Inventor objects.

**2. Split** Removes the top level Separators in both objects. Assumes that all children of the top Separators are also of type SoSeparator. It matches the first child of the source with the first child of the target, etc. If numbers of children in the two objects are not equal, the matching is done for the smaller number of children. The pairs of source-target subobjects will be put through the rest of the pipeline separately until they are to be displayed. At that point, a top level Separator will be added to gather them all.

The motivation for incuding splittin in the pipeline was to allow users to specify somehow which parts of one object should be morphed into what parts of the target object.


The reason why we do it in the manner described above is because we were looking for a method that would be both easy to use and easy to implement.

**Note:** Because of how we split objects into subobjects, all the second level nodes in the input objects HAVE TO BE OF TYPE SoSEPARATOR.

**3. Triangulize** Use the Triangle CallBack supplied by Inventor to split the subobjects into triangles. At that point, we store the objects in our own datastructure.

**4. Scale & Translate** Find the extreme points of each subobject. Then scale and translate it so that it first inside a unit cube. This little bit of pre-processing makes the design and implementation of the morphing algorithm significantly easier.

Information on how much the object was scaled and translated is saved so that later on the object can be restored to its original position and size.

**5.**  At this point the pipeline is stopped until all the subobjects complete the previous steps. As soon as all of them are done we go to the next step, which is...

**6. Morphing!** Yes, this is where an intermediate object is created. The intermediate object is created of a set of triangles. The vertices of these triangles contain their source and target coordinates as well as the coordinates of the current position and other information needed while performing interpolation between the source and the target objects.

The current position of each vertex so that the object looks like the source.

A detailed description of the [morphing algorithm](#) can be found in one of the next chapters.

**7. Convert into an Inventor object** At this point, a Faceset is built out of triangles stored in the intermediate object. The current values of each vertex are used for that purpose. At the end of the process, translation and scaling are applied to each new object so that it appears in the right palce in space. Scaling and translation are interpolation between those of the source and target subobjects.

**8. Display in a SceneViewer window** At this point I results of step 7. for all pairs of subobjects are gather under a common Separator node and displayed in a modified SceneViewer window. The window is augmented with a couple of buttons for directing the interpolation.




**9. If OSR is on => dump an .rgb file** If the Off Screen Renderer flag is on, dump the current contents of the SceneViewer window to an .rgb file so that it can be used for making movies.

More information on the [Off Screen Renderer and the movie making pipeline](#) can be found in one of the next chapters.

**10. One frame forward/backward** Update the current position of each vertex in the intermediate structures corresponding to each of the source-target subobject pairs. The new position corresponds to the next/previous frame in the morphing sequence. The values of the scaling and translating parameters are updated for each intermediate object as well.

**11. Convert into an Inventor object** Same as step 7.

**12.**  Gathering all subobjects under a single Separator.

#### Possibilities for future improvements

- **Fuller handling of material properties.** - At the moment the only material property that we do record and deal with is the `diffuseColor`. It would be desirable to include specular color, transparency, etc. for more pleasing effects.
- **Finding Axis of Least Inertia** - Let us imagine following input to our program: two sticks, one lying on the ground and the other standing up. At the moment, our program would morph the first stick into the second, by shrinking the first one in the X direction and expanding it in the Z direction. It is smooth and acceptable, though it would be much nicer if the stick rotated about the Y axis instead of growing and shrinking. In order to be able to achieve that, our program should be able to find an approximation to the axis of least inertia. We have actually designed a scheme that would handle the problem. The method would use the least squares approximation method to find the best fit line using only X and Y coordinates and another best fit line using only X and Z coordinates of the points. The first line would tell us the angle between the Z axis and the axis of least inertia and the other line would provide the angle between Y axis and the axis of least inertia.

That method is susceptible to errors and is only approximate but we believe it would be a good heuristic to work by. If the method were used, the objects would be rotated, before being scaled, translated and morphed, so that the computed axis of least inertia aligned with the Z axis. The rotation would be undone before displaying the same way it is done with scaling and translation already.

A serious attempt had been made to solve that problem. Our team had acquired a copy of Matcomm software that provides an extensive library of math functions as well as a matlab to C++ compiler. We were not able to make the software fully functional (it would translate but could not make libraries accessible for compilation) without installing it permanently on an SGI workstation. We may work on alternative methods.

## Morphing Algorithms - by Manolis Kamvysselis and Hooman Vassef

In all the approaches we considered, the morphing process required some sort of matching between the triangles in the source and target, then transformations within these matches.

### Explored approaches for the transformations

Consider the case where one polygon (triangle) maps to many. We must somehow create new triangles, and we considered two approaches: "shaping" and "spawning". The first one tries to shape a set of triangles inside the source triangle, the second one spawns all the extra triangles off the edges of the source triangle.

But first, here is a description of how we spawn a vertex from an edge, a process that both approaches use.

Spawning a vertex comes down to creating a zero-area triangle with two vertices shared with an existent edge of the nearest source triangle, and a third vertex at its middle (or some nicer ratio).

Nicer ratio: suppose we need to place point A on an edge BC knowing that the whole will evolve into a triangle A'B'C'. We take the ratio A'B'/A'C', and we set A on BC such that it's the same as AB/AC.

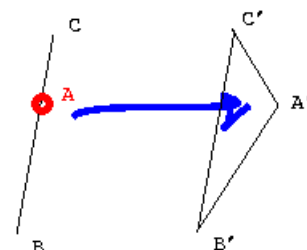


Figure 4: Spawning a vertex

## Shaping (Hooman)

Assume (1) that the "many" triangles form a simply connected mesh: then we can find the external vertices of the mesh. **Figure 5** shows an example of a triangle mapping to a connected mesh with 6 external vertices. We then map the vertices of the source triangle to the three that fit best (using nearest neighbors) of all those external vertices. The other external vertices will spawn off the edges of the source triangle, using the process described above, and marked on Figure 5 by **bright blue** (.....) lines. Now the shape of the mesh is created inside the virtual triangles formed by the vertices from the source triangle and the ones that are spawning from it: the shapes as they are formed at the beginning, when the spawning triangles are still flat, are shown in **dark blue** (.....) lines; the final shapes are in **dark red** (.....).

The challenges in this approach are: to assure assumption (1), that the mesh is simply connected; and to find the exterior vertices in a mesh.

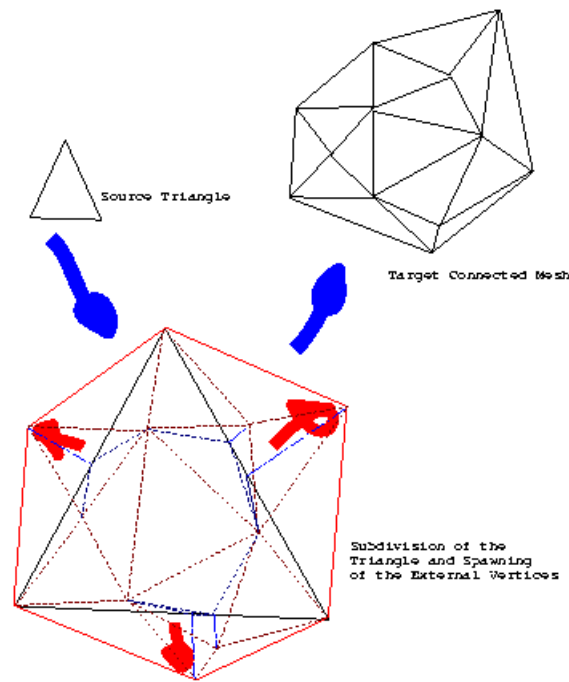


Figure 5: The shaping algorithm.

## Spawning (Manolis)

We have to spawn off vertices on the outsides of the triangles and grow from their edge, as these edges move and morph. Since our program morphs linearly, a connected mesh at the arrival will be connected throughout its travel.

We spawn on the outside so that we don't destroy the "simplicity" of simple surfaces. If we spawn on the inside then we'd have to destroy the top edge from which the new triangles spawned, which cannot be done smoothly since those faces are huge.

The details on spawning (and a colorful figure!) are given later in this paper.

We could, in a more complicated version of the matching algorithm, try spawning off triangles while keeping the entire mesh hole-less. This would mean keeping vertices linked and letting triangles move as their vertices do. However, this involves a radical changing of the entire structure of our program, and we won't show interest in that at this point in time, especially since we can't prove that all polygon meshes are topologically similar, which means, they can be morphed into each other, while all vertices remain shared, a claim which seems easy to disprove by a counter example that will be left as an exercise to the reader.

## Matching and Problems

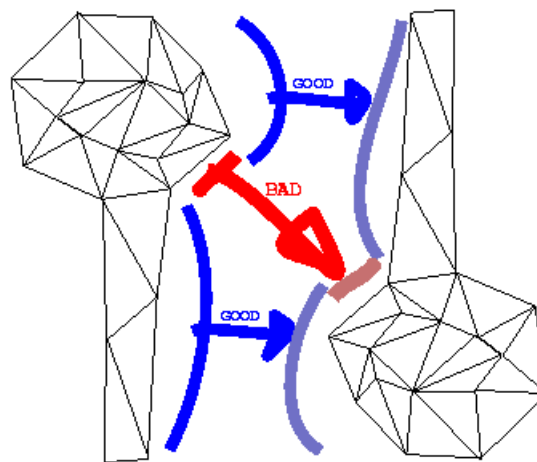
### Matching

The matching process that we first considered would output a list of exclusive one-triangle-to-many mappings that could then be individually processed by one of the algorithms described above. The special property of the "two-way" algorithm is that the one-to-many mappings could go in both directions, from source to target as well as from target to source, solving what Krzysztof called "the lollipop problem":

*The Lollipop Problem:* consider the "lollipop" on **Figure 6**, transforming into an object of similar shape in the opposite orientation: the object has two parts, with high and low triangle concentrations, respectively.

Consider what happens when we use a matching algorithm that maps one-to-many only one way, i.e. the "one" comes from the object with less triangles, the "many" come from the object with more triangles, and if the two objects have the same number of triangles, it's a one-to-one mapping: some triangles in the high density parts of the source object will travel to the low density parts if those map to high density parts in the target object ("BAD" arrow).

Instead we would want the triangles to stay locally, and create or destroy them as necessary ("GOOD" arrows).



**Figure 6:** Spawning a vertex

This two-way matching process works as follows:

- map each triangle in the source object to its nearest neighbor in the target object;
- for each triangle in the target that has not been mapped to, map it the other way to its nearest neighbor in the source object;
- that particular triangle in the source object has been previously mapped to a triangle in the target object: therefore, we must cancel that mapping, and consider if the target triangle still has other antecedents. If not, then map it the other way to the triangle in the source object.

It seems to be quite a brute-force algorithm, but it guarantees a list of one-to-many mappings that solves the lollipop problem. **Note:** one way around the lollipop problem is by forcing the user to create two sub-objects for the lollipop: the candy (dense) and the stick (sparse) separate, and morph them separately into a stick and a candy, respectively.

## Problems

Here are the problems with the approaches considered so far:

- The Shaping algorithm, and to a certain extent the Spawning algorithm (for esthetical results), require a simply connected object. Furthermore, the Shaping algorithm require the "many" to be a simply connected sub-mesh, in the one-to-many mappings returned by the matching process. However, a matching process using nearest neighbors, or any variant thereof, does not guarantee that. In fact, it does not work if you have very long triangles.
- A more serious problem yet, with both the shaping and spawning approaches, is that, using the matching process described above, the vertices that are shared between two sub-meshes should stay together in the morphing process, and that is sometimes impossible.

The problems with the shaping approach seemed impossible to overcome reasonably, and the algorithm itself is very hard to implement to begin with, hence we decided to drop it.

For the final program, we considered a new, more general approach, a more elaborate variant of the spawning process that uses a different matching process, and does not even assume that the objects are connected. It works around the lollipop problem by requiring the user to use separate sub-objects.

## Off-Screen Rendering - by Matt Blum

The morpher program has a feature which allows one to save the current frame as an image file, thus allowing the user to create a string of images which can be put together to make a movie. Open Inventor has a built in feature which saves an Inventor file as a .rgb file, thus allowing one to save still frames easily. It is easy to convert between image formats and Athena has a program that lets one create animated gifs which are little movies with morphing animations.

The off-screen renderer was actually not that difficult to implement because Inventor was nice and provided an OffScreenRenderer library which let me call the following routines to render an rgb file given an Inventor object:

...

```

SoOffscreenRenderer *myRenderer =
    new SoOffscreenRenderer(myViewport);
myRenderer->setBackgroundColor(SbColor(0,0,0)); /* black background */
if (!myRenderer->render(root)) {
    delete myRenderer;
    return FALSE;
}

/* output .rgb file */
FILE *fp = fopen(outputfile, "w");
myRenderer->writeToRGB (fp);
...

```

One little problem I experienced initially with the off-screen renderer was the requirement that the whole scene be under a single Separator node, but since our objects behaved this rule to begin with, it was not a problem. Another small problem with the OSR was that it also initially needed a camera and light position specified in separate files camera.iv and light.iv, but the Inventor libraries in C++ provide routines to dynamically get the camera and lighting information to produce a decent scene. Without this information, the OSR camera position may default to a position pointing away from the object, which would not be very useful.

After getting the OSR to work properly all the time, I incorporated the code as a procedure in the main morphing program and added a button on the main interface that will render the current scene immediately as an rgb file sceneX.rgb where X is the current frame number. The morpher keeps track of the current frame from 1 to N so the frames will be always saved in the correct order.

Thus when the morphing was done, there would be a bunch of files frame1.rgb, frame2.rgb, ..., frameN.rgb which needed to be strung together in a movie. There is a utility on Athena called whirlgif which takes a group of .gif files and makes an animated gif from them (which is easily viewable in Netscape or any other web browser).

The last issue that remained was to convert from .rgb to .gif and there was no converter available. However, doing "man rgb" showed a short 20 line C program on how to read a .rgb file and get its contents. Writing gifs is hard (because it involves a solid knowledge of LZW encoding and a lot of bit manipulation), but writing a .ppm is easy (it is just a string of red, green, blue values) and there is a utility on Athena to convert ppms to gifs, so combining all these tools, I wrote a little script called rgbtogif.

Now that I could make gifs from rgbs, I wrote a short script called makemovie which automatically senses all the files of the form frameX.rgb, converts them into gifs and strings them into an animated gif movie.

## Interface between Open Inventor and Morphing

We want our morphing algorithm to have a natural user-friendly interface and also be interactive, requiring it to do graphics rendering on the fly. Originally, we thought about having a program to take in two sets of triangles each of a specific format because all of our morphing algorithms mapped triangles to triangles. However, if we wanted to have the morpher handle more complicated objects such as cubes, cylinders or spheres, we would have to first manually convert these to triangles to run them through the morpher.

Ideally, we wanted a program that would take objects of a high-level language (such as Inventor format), and morph them interactively. We then realized that in Assignment 4, we used a scan converter that took an OpenInventor file and automatically triangulized it. Using this, it was then easy to put these triangles into our data structures for the morphing algorithm. Since SceneViewer is easily extendible, we could write an extension of SceneViewer by adding a couple extra buttons to handle morphing interactively.

The interface we then decided on was to have the program read in a source and target Inventor files on the command line, bring up a SceneViewer window with added buttons "->" and "<-" to flip forward or backward through the stages of morphing. We set a morphing parameter k that went from 0 to N where k=0 is the source and k=N is the target and the other values represent intermediate scenes. Once all the geometry was calculated at the beginning and the nearest neighbor, all the mappings were completed, then creating intermediate scenes varying k was very fast. In fact, we could have SceneViewer render the intermediate scene on the fly, so as one repeatedly clicked the "->" button, he could see the object morphing continuously.

Later we added another option on the command line, which was the number of frames N or discrete steps k made. Large values made for smoother morphing with more frames. Finally we added a little button to do the off-screen rendering. When one clicks this button, the OSR would render the current scene, keeping track of the index k and rendering an image called "image(k).rgb".

We also included a -debug option on the command line which would print out a verbose report on all the steps the morpher went through so that if it had a problem, the problem could be pinpointed much easier. It is annoying to run a large program and all you see is "segmentation fault" and you have no idea where it broke. The -debug option also shows all the nearest neighbor calculations and how it matched vertices from the source to the target.

## The Algorithm - by Manolis Kamvysselis

We're describing in this part the low-level algorithm to match  $m$  polygons to  $n$  polygons, without preserving any structure. Note that the structure in our structured morpher is obtained by calling this unstructured morphing algorithm iteratively on parts of the scene that the user has specified he/she wants to map to each other, as described earlier in this paper.

We'll also note that we're mapping objects which have been scaled and translated, such that their xyz bounding box is a unit cube centered at the origin.

## Motivation

Two techniques (shaping and spawning) were already presented for creating new polygons. The same techniques apply for making polygons disappear (since one can reverse source and destination environments when doing the matching to guarantee creation instead of deletion of polygons).

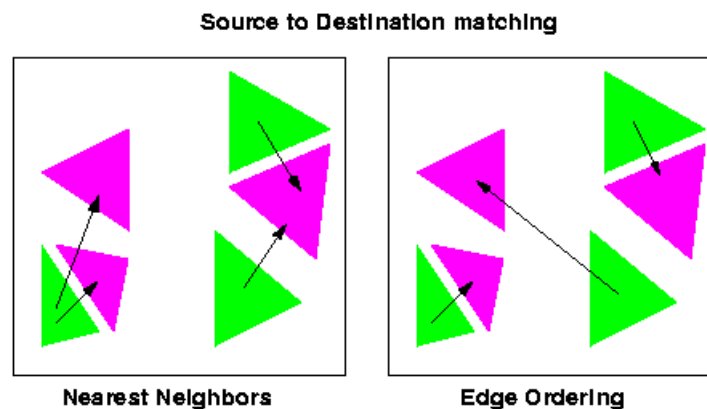
Consider however a morphing algorithm which has no cost assigned to those creations and deletions. In an extreme such case, the program would minimize to zero the old shapes, and create from zero the new shapes. We've taken all intelligence and progressivity out of morphing. We'll try and stay as far away from this as possible.

Thus, our main concern will be minimizing creation and deletion of polygons. Therefore we'll map as many polygons as we can in a one-to-one mapping, and only afterwards will we spawn off the rest as needed.

We'll start by examining the M-to-M mapping first, where  $m$  is the min number of polygons among the source and the target. We'll then introduce the Edge Ordering algorithm for doing such a mapping. Afterwards we'll examine the M to N mapping, and describe how new triangles spawn off in the scene. Finally we'll describe the Incremental Edge Ordering algorithm, an extension to our original Edge Ordering algorithm.

## M to M mapping

We'll use a nearest-neighbor type of algorithm to map the  $m$  objects into each other. A nearest-neighbor algorithm finds for each source the closest triangle in the target environment, and matches the two into each other, as shown in figure 7. In the next part we'll explain how we can obtain the match given by Edge Ordering, but first let's see why a nearest-neighbor approach makes sense.



**Figure 7: Nearest Neighbors and Edge Ordering**

The purpose is to create a one-to-one mapping with no creations and no deletions. Therefore we'll have to map as many as possible, which is what nearest neighbors is doing. Moreover, everything has been scaled to a cube of size 1, and therefore, distances can be used for mapping. Moreover, we don't want to try to be smart in the lowest level of the matching, since we've been smart earlier when matching nodes to nodes. We'll therefore match every polygon to its nearest on the other environment.

**Definition of Nearest.** We're here defining what we mean by distance between two triangles. Instead of the three-dimensional distance of triangle centers, which may seem an intuitive choice for the distance between two triangles, we'll choose the total straight-line distance travelled by all vertices of one polygon to become all vertices of the other polygon in an optimal permutation. In other words, we'll first look through all possible permutations of edges between two triangles ( $\langle 1\ 2\ 3 \rangle$  can match to any of  $\langle 1\ 2\ 3 \rangle$   $\langle 1\ 3\ 2 \rangle$   $\langle 2\ 1\ 3 \rangle$   $\langle 2\ 3\ 1 \rangle$   $\langle 3\ 1\ 2 \rangle$   $\langle 3\ 2\ 1 \rangle$ ), and from those find the one which minimizes length traveled when going from one set of vertices to the other. Save that permutation (a number from 1 to 6) along with the optimal distance, the source and the destination in a data structure we'll call a match, since it's that permutation that we'll use if the two triangles end up matching. Let us add here that from those 6 permutations, three invert the normal and three don't. Therefore, we'll only use  $\langle 1\ 2\ 3 \rangle$   $\langle 2\ 3\ 1 \rangle$   $\langle 3\ 1\ 2 \rangle$ , which don't invert the normal.

More precisely, instead of the sum of those distances, we'll use the sum of their squares. Reader will ask: why can we choose squares instead of distances themselves? Answer: coz 1) we don't have to take the square roots, 2) it gives a more balanced mapping. Illustration: For numbers greater than 1, it works intuitively, since the squares of large numbers are so much bigger than squares of smaller numbers.

Example: sum of 1 10 1 is 12 in sum of distances, and 102 in sum of distances squared. However, 4 4 4 has a sum of distances of also 12, but a sum of squares of 48, which is largely smaller than 102. Therefore, for numbers greater than 1, a more balanced mapping would be chosen. Even though some may find it counter-intuitive, this reasoning also works for numbers which are all smaller than 1 (which is the case in our program, since we're scaling everything to a cube of size 1 when doing the mapping). Example, in the case of .01 .01 and .1, the sum of squares is .0102 while the sum of the numbers is .12. For the same sum .12 we could have .04 .04 and .04, in which case the sum of squares would become .0048, which is smaller than .0102, therefore the balanced thing will still get chosen. A simple argument (convince yourselves, just like we did) is that we can scale everything by 1000 maintaining ratios, and since the scaling is uniform and can be eliminated in the inequality, we're still comparing squares of #'s bigger than 1. We'll therefore use sum of squares that yield more balanced travels.

However, a nearest-neighbor algorithm doesn't guarantee we'll have unique matches for each polygon, as one can see in figure 7. We'll call conflict a match which shares its source or its target with another match. An extended nearest neighbor would work if it has some way of resolving conflicts. It must either either resolve conflicts at the end, which can take a very long time and algorithms for which risk to be very complicated and recurse indefinitely, or if it can resolve conflicts incrementally, by mapping to the nearest polygon which is not already matched with another, in which case the algorithm is partial since it will match the nodes visited first without guaranteeing a globally optimal matching.

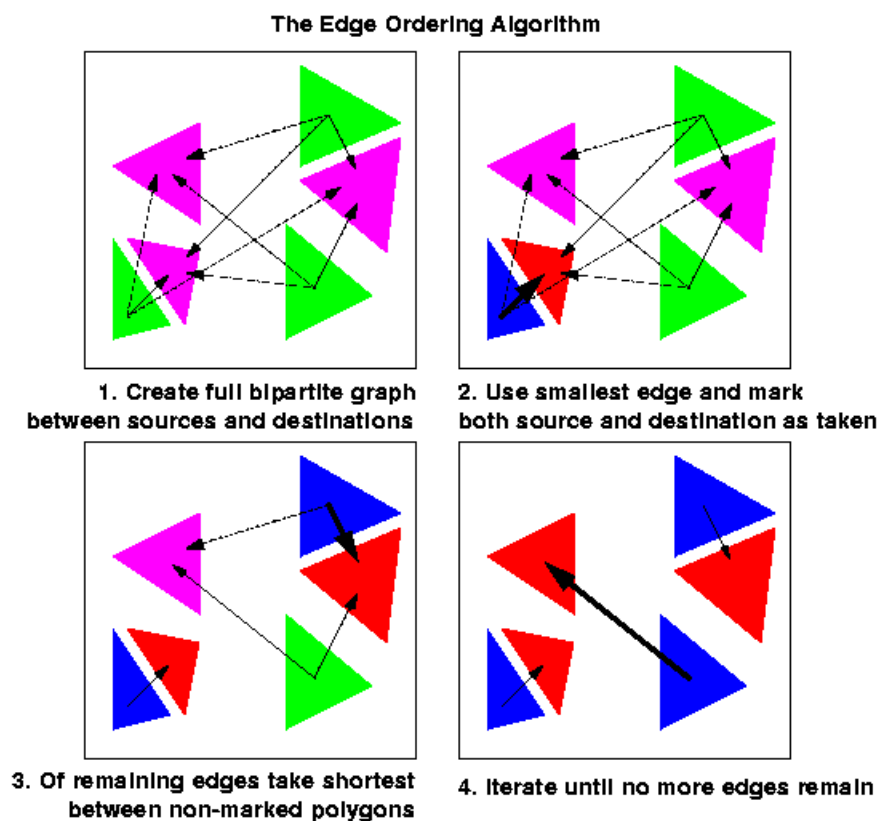
## Edge Ordering

An alternative to a nearest-neighbor algorithm is the Edge Ordering Algorithm, introduced in 1997 by Manolis Kamvysselis, a junior in computer science at MIT. He originally used it for his final project in Computer Graphics, and we'll use it here, since it seems fit.

In this algorithm, each triangle is a node, and each matching between two triangles is an edge between two nodes. Since matchings are done only between the two non-intersecting spanning sets of sources and targets, the matching problem becomes the construction of a bipartite graph which connects all nodes, and each node only once.

The algorithm, illustrated in figure 8, is as follows:

- First construct a full weighted bipartite graph connecting every source to every target, with the weight here being the distance between two polygons, as defined above. This takes  $m^2$  time.
- Then order the edges (hence the name Edge Ordering) using your favorite sorting algorithm (even bucket sort would work here, since we know all objects are inside a unit cube). My favorite one is quick sort, which like all optimal comparison sorts, takes  $n \lg n$  average time. The number of edges being  $m^2$ , this step takes  $m^2 \lg m$  time.
- For every edge, in the sorted list of edges, if both parents are free, then make a match, otherwise, skip the edge, and take the next one. This takes  $m^2$  time, since we have to look at all the edges. It makes however, only  $m$  matches are made, if we want to use the number of matches made as our cost.





**Figure 7: The Edge Ordering Algorithm**

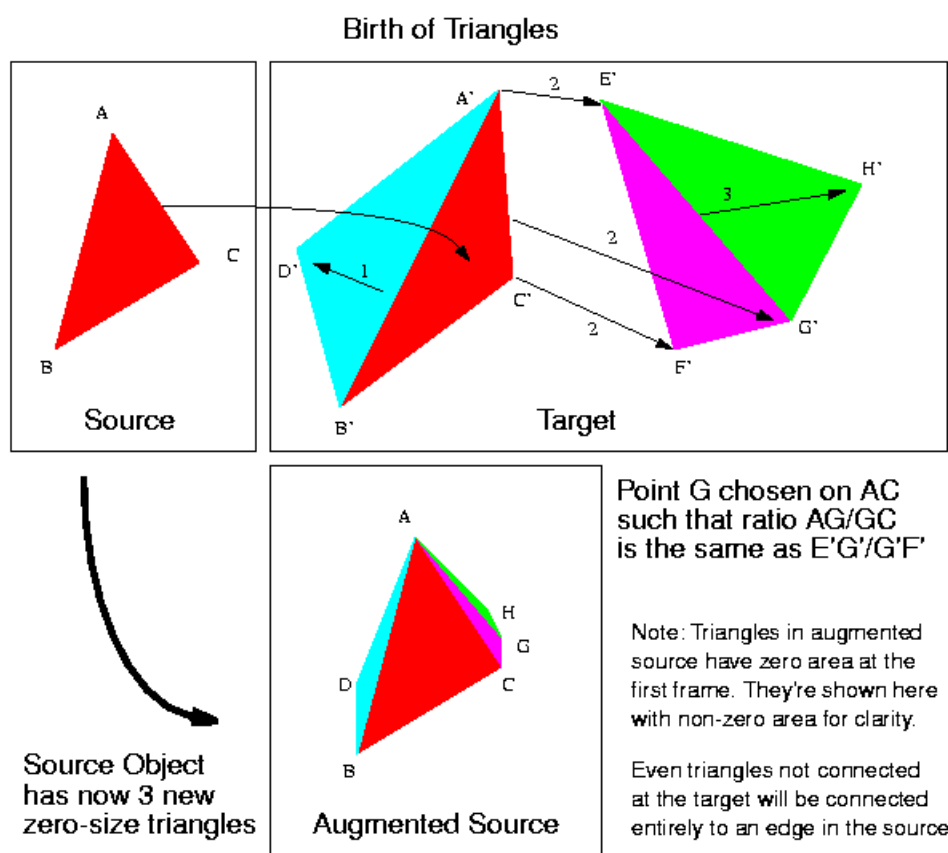
**Correctness:** When no more edges remain in our sorted list, we'll have achieved our bipartite graph. Proof: suppose at least one pair is unmatched. since our list contained an edge for every pair of nodes, it must have either not seen their edge (therefore we're not done, contradiction), or seen their edge and didn't connect them (therefore, one of the nodes must have been taken, therefore no unmatched pair of nodes exist).

Greedy strategy is optimal in polynomial time. The problem of finding the minimum length cycle in the full bipartite graph is NP hard, therefore, we can't wish to achieve an optimally optimal solution. However, our greedy approach does pretty well for its running time, since at each step it's adding the minimum weight edge available.

## M to N mapping

We assume we've already mapped  $m$  polygons to  $m$  others, and there remain  $n-m$  of them unmatched all in the same environment (by correctness argument) either in the source or in the target environment, without loss of generality.

We now have to spawn off additional triangles which have zero area in the source environment, but which will become good and healthy triangles in the target environment.

**Figure 8: Birth of Triangles**

As illustrated in figure 8, for each unmatched triangle in the target environment, we'll look through the list of currently matched triangles in the target environment (the same environment this time), and we'll find the nearest edge from which to spawn the unmatched triangle. We'll then create a zero-area triangle on the corresponding edge of the corresponding triangle in the source environment (the edge from the source that'll become the best-birth edge in the target). The two endpoints will correspond to two endpoints, and the third will be created in the middle preserving the edge ratio described in the figure.

**Measure of nearest.** This time nearest edge is the one that minimizes the total length to be traveled from the target to the target were we to displace the triangle only in the target. The reason is that we want to keep a kind-of constant flow of triangles, and therefore we'll trust that the edge from which the triangle is spawned is already traveling right and doing the right thing, and we'll spawn off a triangle which will travel the least compared to that edge. It will travel the least, since it's starting on the edge, and it's finishing the closest to it than all other triangles. We now have to check more permutations for each triangle to be mapped. We'll also keep more information. We'll remember the parent triangle from which the unmatched triangle will spawn, the edge on that triangle from which it will spawn, the edge of the child triangle that will correspond to that edge, and the ratio of the two other edges. All that information will be kept in a structure

we'll call a birth.

## Incremental Edge Ordering Algorithm

To do the  $m$  to  $n$  matching, we'll therefore use the following algorithm, illustrated in figure 9.

- From all possible births, choose the one which results to the with minimum distance to travel.
- Create the match of the child on corresponding edge in source environment.
- Update list of births eliminating all those giving birth to the newly matched polygon, and adding all those which can start from the newly matched polygon to a currently unmatched polygon.
- Iterate until there are no more births to consider

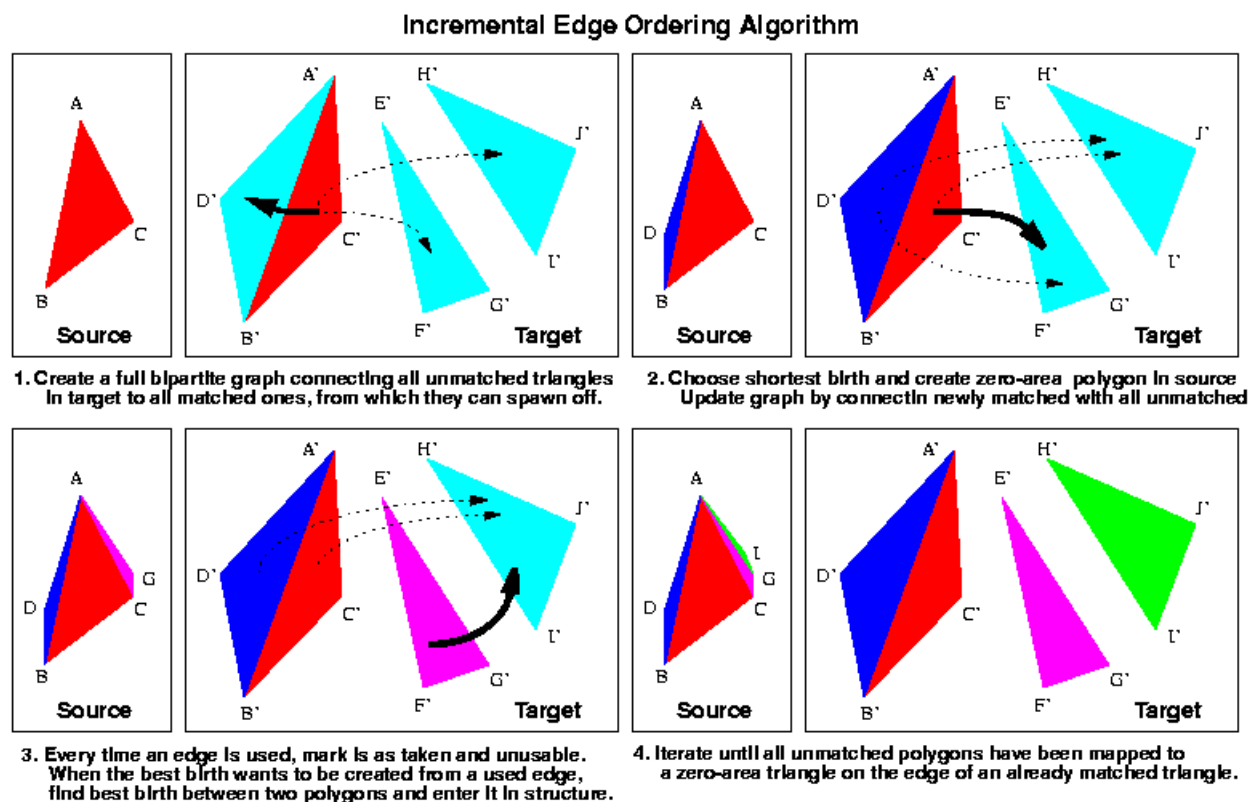


Figure 9: Incremental Edge Ordering Algorithm

This algorithm sounds awefully expensive. We can't use the Kamvysselis Edge Ordering Algorithm, since it assumes the list of best edges does not get updated. It sorts it once, and keeps it so forever. We therefore would have to do an insertion sort if our data structure is a list.

To the rescue comes the Incremental Edge Ordering Algorithm, also developped in 1997 by Manolis Kamvysselis for his graphics final project. Same allegories, but this time we're sorting incrementally, hence the name Incremental Sorting Algorithm.

We'll keep a data structure containing the possible birth to be made between all matched triangles (and the best edge to use and the distance achieved by using the best edge), and all unmatched triangles.

The way to achieve such a functionality with an optimal time is a heap, a birth heap we'll call bheap. We insert in  $\lg n$  time, and we extract the min in  $\lg n$  time. The heap will contain all possible births between all matched polygons and all unmatched polygons.

The other problem is that the data inside our data structure is changing. When an edge is used to create one polygon, it cannot be used to create another. Therefore, if two unmatched polygons both had selected the same edge to be created from, as soon as one is selected, the other best birth will be invalid. It would be wrong to update every single best birth involving a polygon every time a birth is used with that polygon. Instead, we'll take care of such cases when trying to use a birth. If the mother edge is taken, we'll call best\_birth again, with the same two polygons, only this time, it will consider the edge taken. When the match is made, we'll insert the node again in the heap.

### The Incremental Edge Ordering Algorithm

- For every unmatched polygon  $i$ 
  - For every matched polygon  $j$ 
    - Insert best\_birth( $i, j$ ) into the bheap.



- Until no more polygons are unmatched (  $O(n-m)$  )
  1. Choose the birth with the smallest dist. (  $O(\lg((n+m)*(n-m)))$  )
  2. If the child polygon is already matched, skip. loop.
  3. If the parent edge is taken, call `best_birth` on parent and child and insert birth into data structure, loop.
  4. Otherwise make birth happen. Mark the unmatched polygon as matched, and find `best_births` for each of the remaining unmatched polygons and insert them into the births heap. (  $O(m * \lg((n+m)*(n-m)))$  )
  5. Mark the edge that gave birth as taken. Mark the edge that was born as also taken. The other two edges of new polygon can give birth to unmatched polygons also.

Correctness: at any time, at least all possible births are included in the data structure, and the optimal one is handed in by the call to `bheap_return_min`. No impossible birth will be made between two polygons, since we're checking all criteria that would disallow the match. If, when the edge of a parent is already used, we're reconstructing a birth among those two polygons, we're guaranteed that it won't be better than any birth already used, since it's not better than the previous birth between those two polygons (otherwise, it would have been selected in the first time around), and the impossible birth is no better than any of the ones already used otherwise, it would have been extracted from the bheap earlier. Putting the new reconstructed birth and back, guarantees that if the birth leads to an optimal combination later on, it will get selected.

Greediness: by arguments above, at each step we're making the optimal choice for a birth. Finding the one best combination is however NP hard, and we're doing pretty well for a polygomial (times logarithmic) time.

## Possibilities for future improvements

An interesting extension to our three-dimensional morpher could be a hybrid between two and three-dimensional morphing by also including morphing of texture maps between objects. In most applications, scenes are rendered as simple objects but with complicated textures and morphing of texture maps would help to make the transition appear even more realistic.

## Lessons Learned or How We Benefited From Doing This Project

- **Open Inventor** - Through working on this project we had a chance to become intimately familiar with the Open Inventor. The first discovery that we made about it was that Open Inventor was not a closed program or system but an extensive collection of tools accessible to programmers.
- **Morphing** - We researched the field and learned about many interesting things that other people are doing. Unfortunately, we did not find any papers that were directly applicable to our problem. On the other hand, that allowed us to build an entirely new and entirely our own solution! Very rewarding. Let's get some patents on those nice algorithms! Free ticket to SIGGRAPH '99 after graduation :o)
- **Software Engineering** - For most of us it was a second team project in software development. We learned a lot about how a team should work and what are the efficient and inefficient ways of managing code and work.

## Individual contributions (division of labor)

- **Matt** - Off Screen Renderer, parsing command line arguments, movie-making pipeline, the 2D morphing demo for our front page.
- **Manolis** - Morphing - Matching - Edge Ordering Algorithm - Incremental Edge Ordering Algorithm
- **Hooman** - Morphing
- **Krzysztof** - Everything else, i.e. all of the pipeline described above except for the morphing and OSR.

## Acknowledgements

The team would like to thank **Jonathan Levene**, the god of Inventor, who was a non-exhaustable source of inspiration in our struggles with Open Inventor.

The team would also like to express its gratitude towards our project supervisor, **Luke Weisman**, for his patience (when needed), impatience (when nothing else worked), and cool ideas.

We would also like to thank our **friends, brothers, sisters, parents, roommates, pets, stuffed animals**, and last but not least the **TA's of other classes** too, who supported our efforts of spending endless hours on this project. Thanks!

Finally, the team would like to thank the **Anonymous Genius** who wrote the IvScan. A lot of our knowledge about the Open Inventor as well as numerous parts of our code come from his work.

## Bibliography

*The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2; SGI On-Line edition*

---

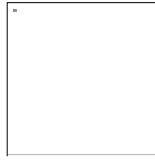
## The Team: **MaD-HooK** [morph@mit.edu](mailto:morph@mit.edu)



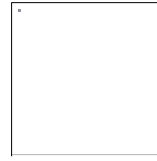
[Manolis Kamvysselis](#)  
[manoli](#)



[Matt D. Blum](#)  
[mdblum](#)



[Hooman Vassef](#)  
[hvassef](#)



[Krzysztof Gajos](#)  
[kgajos](#)

Final Project for [6.837, Computer Graphics](#). Professor: [Seth Teller](#) ([seth@graphics.lcs.mit.edu](mailto:seth@graphics.lcs.mit.edu)) TA: [Luke Weisman](#) ([luke@graphics.lcs.mit.edu](mailto:luke@graphics.lcs.mit.edu))



# RoboLogo

```
DO.UNLESS [
  DO.WHILE [ TRUE
] READSENSOR FR [
  PRINT "Front Right Sensor Hit!"
]
```

## RoboLogo: Teaching Children how to program Interactive Robots

[Manolis Kamvysselis](#) - [Jeremy Lueck](#) - [Christopher Rohrs](#)  
[manoli@mit.edu](mailto:manoli@mit.edu) - [jlueck@mit.edu](mailto:jlueck@mit.edu) - [chrohrs@mit.edu](mailto:chrohrs@mit.edu)

[Pictures](#) of the truck  
[Code](#) examples

## I. Introduction

### a. RoboLogo Overview

RoboLogo is a system that enables children to program interactive robots. Children can program a robotic truck that interacts with the environment without having to deal with low-level implementation details.

Using iLogo, a high-level interactive language, the children can easily describe the robot's actions in the environment and just as easily program the robot's reactions to external events, such as encountering an obstacle. For example, to program a robot that bounces between two walls, the user need only write:

```
do.while [
  do.unless [
    forward
  ] (or readsensor fl readsensor fr) [
    # do nothing
  ]
  do.unless [
    backward
  ] (or readsensor bl readsensor br) [
    # do nothing
  ]
] TRUE
```

This iLogo program is parsed and translated to assembly code. The resulting A51 file is linked with the low-level RoboLogo library routines and compiled into machine code. The Intel HEX file thus generated is downloaded onto the RoboLogo truck via the serial port. Unaware of the details, the child can now disconnect the programmed robot watch it run, and interact with it.

The goal of our 6.115 project was to enable this transition from the abstract description of the robot's behavior to an actual moving robot. This involved managing the mechanics and sensors of a robotic truck, laying out the electronics to control it, writing the assembly routines that coordinate its actions and sensors, developing the iLogo language, and implementing the compiler that translates iLogo to assembly code.

### b. On the footsteps of LOGO

Like the name suggests, RoboLogo is based on the original LOGO language, developed in the 1960's at Bolt, Beranek and Newman, Inc. LOGO originally allowed children to program a small robot-called the ``turtle''-to move around on the floor in different patterns. Eventually the robot moved from the floor to the computer screen, and children could program the turtle to draw colorful, pretty pictures.

As an example, here is the LOGO code to draw a series of rotated squares:

```
REPEAT 36
  GO SQUARE
  RIGHT 10
NEXT
END

# SQUARE
```

```
REPEAT 4
  DRAW 100
  RIGHT 90
NEXT
RETURN
```

LOGO is designed to be easy to learn, yet powerful enough to express complex ideas. It has a surprisingly rich set of features, including variables, procedures, and looping constructs. LOGO is an excellent tool for teaching children about analytic thinking and programming, yet it is enjoyable and rewarding.

The limitation of LOGO however is the lack of feedback from the environment. There is no way of expressing an event occurring in the outside world. If the robot bumps into a wall, it won't react. Even if someone gets on its way, a LOGO robot will keep going.

### **c. Interactive LOGO**

Setting out to overcome this limitation, we developed iLogo, an interactive version of LOGO. Simple constructs in iLogo extend the original LOGO language with interactivity capabilities of reading sensors and transferring control to different parts of the program.

We went back to the original mechanical idea of the LOGO turtle. However, our truck differs in one critical way; it has sensors that allow it to respond to its environment. Children may read and respond to sensory inputs without the complexity of interrupt vectors, AD converters, and register banks.

A child can thus program the truck to interact with the environment, bounce between two walls, follow a human, or even navigate a maze. The result should become an amusing but instructive toy.

### **d. Report Overview**

Our report will present the RoboLogo architecture in a bottom up fashion. We will first talk about the low-level issues of making the robot work: basic hardware, sensors, controls. Then we will treat the language implementation: structuring the grammar, writing the compiler, translating iLogo.

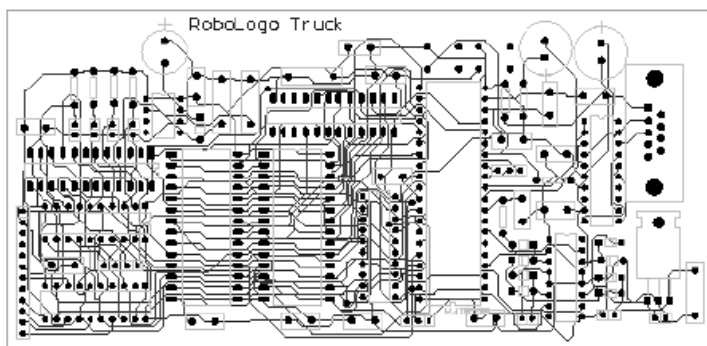
## **II. Robotic Truck**

In this section, we describe the mechanics of our truck, and the circuit and assembly primitives to control it. Because hardware and software are so closely linked, we will treat those two aspects jointly.

### **a. Basic Hardware**

Our project started with the chassis of a toy remote-control truck. This truck has a driving motor which actuates all four wheels, a steering motor that changes the direction of the front wheels, and a simple feedback system with which one can determine the position of the front wheels. The motors are powered by a 9.6V nickel-cadmium rechargeable battery. On the four corners of the truck we mounted small switches extended with metal arms to act like bump sensors.

To control this truck we laid out a custom-made, two-layer printed circuit board. The schematic for this board is shown in Figure [\[1\]](#). The layout of the board is shown in Figure [\[2\]](#). This circuit board is basically the 6.115 microprocessor system. At its heart is an Intel 8051 microprocessor connected to a ROM, RAM, and serial port. An LM7805 step-down regulator converts the 9.6V power from the truck's battery to a 5V logic supply. An LM18293 push-pull motor driver connects the programmable counter array (PCA) of the 8051 to the truck's motors. In addition, the circuit board has space for an AD converter, a -5V switching power regulator, and an additional LS244. Unfortunately, we did not have time to use these parts.



**Figure:** RoboLogo Printed Circuit Board

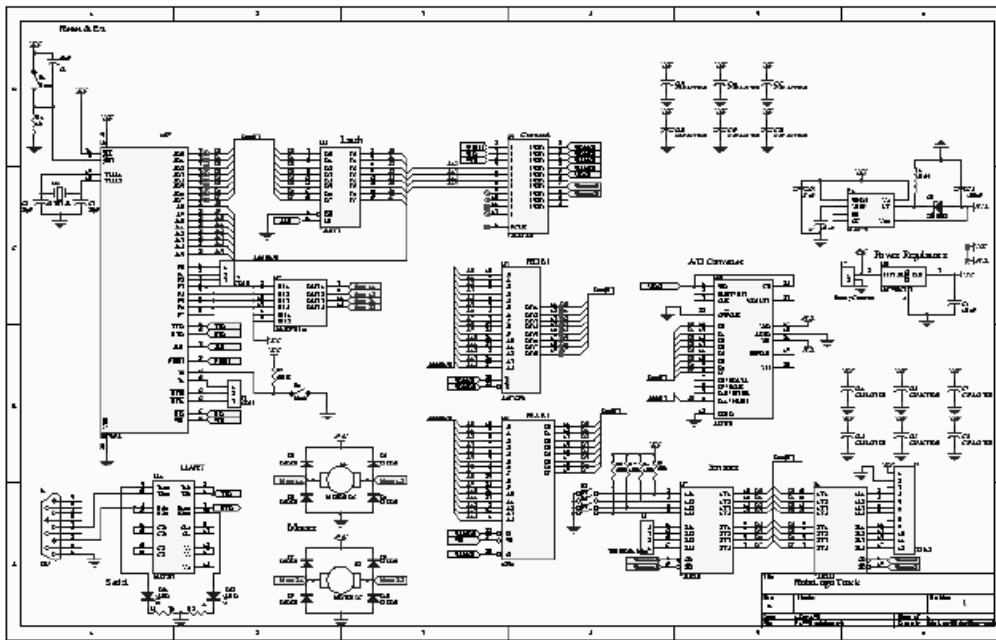



Figure: RoboLogo Schematic

## b. Sensors

The external switches (the steering box and bump sensors) are connected with pullup resistors to the input of an LS244 tri-state driver. The output is connected to the processor's data bus. A PAL selectively enables the output of the LS244, as well as the RAM, ROM, unused AD converter, and second LS244. The resulting memory map is similar to that of the 6.115 circuit, except that low data memory is divided between the AD converter and the tristate drivers. The program for the PAL that controls the memory chips is shown in Figure .

```
device 22v10;

positive PSEN@2, RD@3, WR@4, A12@5, A13@6, A14@7, A15@8;
negative RomCS@23, RomOE@22, RamCS@21, RamOE@20, AdCS@19,
    Sensors1E@17, Sensors2E@16;


RomCS = 1;
RomOE = /A15*/PSEN;

RamCS = 1;
RamOE = A15*/RD + A15*/PSEN;

AdCS = /RD*/A13*/A14*/A15;

Sensors1E = /RD*/A13*/A14*/A15;
Sensors2E = /RD*/A13*/A14*/A15;
```

Figure: Program for memory pal

As mentioned earlier, sensors are memory mapped. Reading from external memory locations 0x2000 through 0x2FFF causes a "sensor word" to be written to the bus. The truth table for the sensor word is shown in Figure . Note that bit 4 is unused. Also note that the logic for wheel positioning is somewhat complicated; this is a direct result of the implementation of the truck's steering box.

Sensor Index	7	6	5	4	3	2	1	0
Front right sensor pressed	X	X	X	X	X	X	X	0
Front left sensor pressed	X	X	X	X	X	X	0	X
Back right sensor pressed	X	X	X	X	X	0	X	X
Back left sensor pressed	X	X	X	X	0	X	X	X
Wheels centered	0	X	X	X	X	X	X	X
Wheels completely right	1	0	X	X	X	X	X	X
Wheels completely left	1	X	0	X	X	X	X	X
Bit Meaning	TS	TR	TL		BL	BR	FL	FR

Figure: Truth table for sensor word

The procedure `ReadSensor` in `sensors.a51` is a convenient way of reading the sensor values. This file is shown at the end of the report. Since none of the switches are hardware debounced, software must be careful when using `ReadSensor` frequently.

## c. Controls

It is easy to drive the truck forward and backward; one simply sets the duty cycle of the pulse-width modulated waveform (PWM) from the 8051's PCA modules 0 and 1. Likewise, one can turn the steering wheels completely left or right by setting the duty cycle of PCA modules 2 and 3. Centering the wheels is trickier, as one must use the wheel position sensors. Unfortunately, our truck does not provide enough feedback to accurately turn the wheels partially, say to 30 degrees.

With the above primitives, it is fairly easy to construct higher level routines for driving the truck. The procedures `ForwardStraight`, `ForwardLeft`, `ForwardRight`, `BackwardStraight`, `BackwardLeft`, `BackwardRight` drive the truck in the specified direction forever, and do not return. Obviously these procedures are only useful in a context where they may be interrupted. The procedures `ForwardStraightTG`, `ForwardLeftTG`, `ForwardRightTG`, `BackwardStraightTG`, `BackwardLeftTG`, `BackwardRightTG` drive the truck in the specified direction for a given time and speed. A few implementation notes are in order:

- All of the procedures use a carefully calibrated busy loop, `WaitSec`, to time the motors. Unfortunately, if interrupts occur, `WaitSec` is no longer accurate. A better solution is to use a timer to count the number of cycles the truck has driven for.
- All of the procedures take a discrete ``gear" to specify the speed. The reasons for this are two-fold; first, by limiting the power of the truck, we simplify the interface to children. Secondly, it allows use to calibrate the ``gears" so that, for example, 10 seconds forward in first gear is the same distance as 10 seconds backward in first gear. This is necessary because our truck goes slower in reverse direction for mechanical reasons.
- All of the above procedures turn the steering wheels before-not during-moving forward or backwards.
- It is necessary to continuously apply power to the steering wheels when moving in the forward or reverse directions. Otherwise, the truck will not turn sharply.

## III. iLOGO

When trying to develop the Interactive Logo language (iLogo), we wanted the language to possess a few key features:


1. Easy to learn - The language must be fairly intuitive to users who are not necessarily familiar to programming.
2. Easy to implement - The language must be compact enough for us to be able to implement in the short amount of time necessary to complete this project
3. Capabilities for reacting to stimuli - The language must have primitives which allow the user of the language to write programs which easily transfer control based upon outside stimuli, in this case sensors on the truck.

The result of our development was a subset of the Berkeley Logo language. We took the Berkeley Logo language design as our base and then added a primitive for reading sensor and an exception-based control structure. We also decided to take out a lot of unneeded structures and primitives to scale the down the implementation of the language. We decided to use Berkeley Logo because it is freely distributed on the web and comes with a user manual which was the closest free specification of the Logo language we could find. It also helped us accomplish key features 1 & 2, because this version of logo is already known for being easy to use and the code which came with the Berkeley Logo package already had a scanner for the language. Berkeley Logo can be found at:

<ftp://cher.media.mit.edu/pub/logo/software/ucblogo/>

Unfortunately, adding any new structures to the existing scanner proved very difficult, as the scanner was hand coded and contained little if any documentation. Fortunately, there are other free distributed tools on the web which helped us in the compiler generation. These tools will be discussed in the compiler implementation section.

## a. Grammar

After splicing out the unnecessary parts of Berkeley Logo and adding our two new features, we formalized the context free grammar for the iLogo language, shown in Figure .

```

Program ::= Command*

Command ::=
    MakeCommand
    | IfCommand
    | WaitCommand
    | ForCommand
    | DoWhileCommand
    | DoUnlessCommand
    | MoveCommand
    | PrintCommand

```

```

MakeCommand ::= 'MAKE' Identifier Expression

IfCommand ::=  'IF' BooleanExpression InstructionList [InstructionList]?

WaitCommand ::= 'WAIT' NumericExpression

ForCommand ::= 'FOR' List InstructionList

DoWhileCommand ::= 'DO.WHILE' InstructionList BooleanExpression

DoUnlessCommand ::= 'DO.UNLESS' InstructionList
                  BooleanExpression InstructionList

MoveCommand ::= MoveOp NumericExpression NumericExpression

PrintCommand ::= 'PRINT' String

InstructionList ::= '[' Command* ']'

Expression ::=  BooleanExpression
                | NumericExpression
                | List
                | Identifier

BooleanExpression ::=  '(' BooleanOp Expression Expression ')'
                    | '(' LogicalOp BooleanExpression+ ')'
                    | Boolean
                    | 'READSENSOR' Sensors

NumericExpression ::=  '(' NumericOp Expression Expression* ')'
                    | '(' NumericExpression ')'
                    | Integer

Sensors ::= 'FR' | 'FL' | 'BR' | 'BL'

MoveOp ::= 'FORWARD' | 'FORWARDR' | 'FORWARDL' |
           'BACKWARD' | 'BACKWARDR' | 'BACKWARDL'

BooleanOp ::= '=' | '>' | '<' | '>=<' | '>='

LogicalOp ::= 'AND' | 'OR' | 'NOT'

NumericOp ::= '+' | '-' | '*' | '/' | '%'

List ::= '[' Expression* ']'

Boolean ::= 'TRUE' | 'FALSE'

Integer ::= 0, 1, ..., 255

Identifier ::= [a-z|A-Z][a-z|A-Z|0-9|_]*

String ::= '"' ~["]* '"'

```

**Figure:** iLOGO grammar

This grammar describes an iLogo program as a list of commands. A command can be either a MakeCommand, IfCommand, WaitCommand, ForCommand, DoWhileCommand, DoUnlessCommand, MoveCommand, or PrintCommand. A MakeCommand is much like an equate statement in many language. It equates a value to some identifier. The wait command is a primitive which causes a delay based upon the result of the expression. A ForCommand is exact like a most 'for' loops. It takes an list which is an identifier, a lower bound, an upper bound, and an optional scale between iterations. The DoWhile Command is also much like a standard 'do-while' loop. It causes a loop until the a boolean condition is no longer true. The DoUnlessCommand was added as an exception-style control structure which executes a set of commands unless an exception condition is met. The subtleties of this new DoUnlessCommand are described in detail in section DoUnlessCommand. The MoveCommand is a primitive for moving the truck in a direction for a specified time and speed. Finally, the PrintCommand is a primitive for generating diagnostic messages from the truck over the serial port. We decided that these primitives were simple, but powerful enough to possibly create larger primitives written in iLogo (such as turning 90 degrees or turning in place).

Expressions in iLogo come in two varieties, numeric and boolean. This gives iLogo expressions only two types, integer and boolean. Expressions are created using Scheme-type parentheses. This approach was used in the earliest version of Logo, when Lisp and Scheme were the hot languages in the computer science field. Using this form of expressions make the parse trees for expressions explicit, and disallows operator precedence which complicates parsing and language usage.

## **b. Implementation**

Creating a grammar like the one described above was necessary for using freely distributed compiler tools. These grammars are used as input for tools like Lex and Yacc to create custom lexers and parsers for the languages described by the grammar written in C.



Fortunately, we were aware of a nifty tool developed by Sun for the Java language. We decided to use the JavaCC/JJTree tools created by Sun for generating a custom parser for our iLogo language written in Java. Writing our compiler in Java was a big win for us because we could develop the compiler on multiple platforms (UNIX, Linux, Win32) and it would also link well with the visual layer we were hoping to develop for the language later in the project. It was also a big win to use JavaCC/JJTree because we had experience using it, and the turnover rate for creating a compiler using these tools was the smallest. JavaCC and JJTree are freely distributed Java programs, and can be found at:

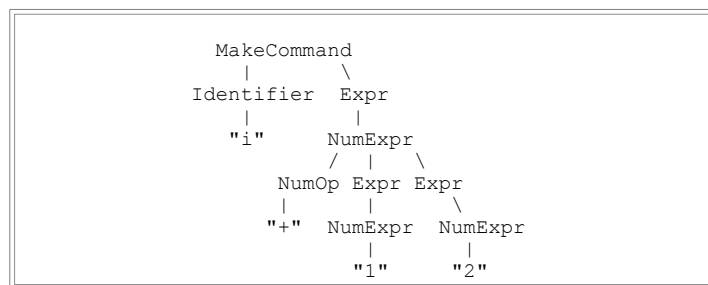
<http://www.sun.com/suntest/products/JavaCC/index.html>

The result of using JavaCC/JJTree with our grammar was about 4000 lines of Java code which would lex and parse any iLogo program specified by our grammar. All that we would need to add to this would be actions for converting a parse tree into translated a51 code.

Translation of iLogo into a51 assembly takes place in three separate stages:

1. Parsing - verifying proper syntax for an iLogo program. This is done for us by the code created by JavaCC/JJTree
2. Static Checking - checking the semantics of the input program for errors which slip through the parser (e.g. checking limits on integer constants, checking the types of arithmetic expressions for integers, etc.)
3. Translation - translates each separate chunk of an iLogo program into it's appropriate a51 form. This translations follows a set of known rules.

After the parsing stage, the text version of the input program is changed into a parse tree, which is traversed once each by stages 2 and 3. (Of you are unfamiliar with parse trees, see Figure [4](#).)



**Figure:** Parse Tree for expression MAKE i (+ 2 3)

We were lucky enough to know that every successfully checked program would be translated, so extra information is stored in the parse tree to speed up the translation (such as type information for identifiers, max depth for expression, and a collection of string constants used).

The translation stage followed some very simple rules for performing a correct a51 translation, albeit not very efficient. We were not striving for efficiency, but could easily improve the run-time behavior of the output a51 program by adding other optimization phases to the compiler. The rules for the translator was as follows:

1. All integers identifiers are 1 byte, unsigned words
2. All boolean identifiers are bits
3. All identifiers have their own memory location (no overlap or register usage for identifiers)
4. All expressions leave their values in the accumulator, in the case of boolean expressions, the value is left in ACC.7
5. All nested expressions leave temporary values in a space assigned for nested expressions. The size of this memory area is determined by the maximum depth of expressions with the entire program
6. All boolean operations are short-circuit
7. All iLogo command primitives have associated iLogo a51 library routines (e.g. ForwardStraight, Stop, etc.)

After the translation stage is completed, the output assembly file is assembled and linked with the library routines and a special startup file to create the output 8051-specific program. This hex version of this program is downloadable to the truck.

### c. DoUnlessCommand

These above rules handle all of the commands and expressions of the iLogo language except for the DoUnlessCommand. This command will execute a list of commands unless a boolean condition is met. If so, control is switched to a new list of commands for handling the exception condition. Figure [5](#) shows an example DoUnlessCommand which loops forever until the front right sensor is hit, and then prints a diagnostic message:





```

DO.UNLESS [
    DO.WHILE [] TRUE
] READSSENSOR FR [
    PRINT "Front Right Sensor Hit!"
]

```

**Figure:** simple.il: example DO.UNLESS command

startup.a51 starts a polling loop before it calls the compiled iLOGO program. At the start of a DO.UNLESS block, the compiled code enables interrupts and installs the address of the compiled UNLESS handler in `_Handler`. It also saves the stack pointer in `_SpSave`; we will see why shortly. When enabled, timer1 generates an interrupt every 255 cycles, which is handled by `_PollHandler`. After saving a few registers, `_PollHandler` unconditionally jumps to the address in `_Handler`.

The generated handler must first switch register banks before doing any work. Then it evaluates the boolean expression specified in the UNLESS clause. If the condition is false, the handler restores registers and returns normally from the interrupt, allowing the DO block to continue executing. However, if the condition is true, the handler aborts the DO block and begins executing the UNLESS block. This requires a bit of trickery. First, the handler calls `WheelOff` and `Stop` to turn off the turning and driving motors. Secondly, the handler restores the stack to the position at the start of the DO block, thus terminating any procedures that might be executing at the time of the interrupt. Finally, it pushes the address of the UNLESS handler onto the stack and calls `RETI`, thus exiting from the interrupt and passing control to the UNLESS handler. The result of translating the code in Figure  is shown in Figure , with comments.

```

;;;;;;;;;; DO block ;;;;;;;;;;;
MOV     _SpSave, SP           ;save stack
MOV     _Handler, #HIGH(_H0) ;install handler
MOV     _Handler+1, #LOW(_H0)
MOV     TL0, #0              ;enable polling
SETB    ET0
SETB    TR0

_L2:
MOV     A, #1                ;body of DO block
JZ      _L3
LJMP    _L2

_L3:
LJMP    _L0

;;;;;;;;;;Handler for DO block;;;;;;;;;;
USING 1
_H0:
PUSH    PSW                  ;save state (some saved by
MOV     PSW, #(1 SHL 3)      ;by _PollHandler in startup.a51)
PUSH    B
MOV     R3, #FR              ;test boolean condition
CALL    ReadSensor
JZ      _L1
CLR     ET0                  ;if true, 1. turn off timers
CLR     TR0
CALL    Stop                 ;          2. stop wheels
CALL    WheelOff
MOV     SP, _SpSave          ;          3. restores stack to
MOV     DPTR, #_L0           ;          start of DO block
PUSH    DPL                  ;          4. pass control to
PUSH    DPH                  ;          UNLESS block
RETI

_L1:
POP     B                    ;if false, restore state
POP     PSW                  ;(some saved by _PollHandler)
POP     ACC                  ;and return from interrupt
POP     DPH
POP     DPL
RETI

;;;;;;;;;;UNLESS block;;;;;;;;;;
_L0:
MOV     DPTR, #_S0
CALL    PutStr
CALL    Stop
CALL    WheelOff
RET

```



**Figure:** simple.a51: translated version of simple.il, with comments

## **IV. Discussion**

### **a. Separation of Labor**

Manolis was mainly responsible for the low-level routines, that controled moving back and forth, turning the wheels, reading individual sensors, waiting for a number of seconds. He was also responsible for the mechanical aspects of the truck, such as motor speeds and power, mounting switches and bump sensors and reverse engineering the turning control wires, that give feedback on wheel position. Having taken 6.270 and having had experience building robots before 6.115, we was best suited for this job.

Jeremy was in charge of the compiler. He wrote a parser, interpreter, compiler and translator. He found the Berkeley LOGO specification, the javacc tool for building parsers, and he wrote two thousand lines of Java code to make iLOGO a reality really early in the project. This is the fourth compiler Jeremy has written, and he was able to complete his part of the project in time, being already familiar with the development tools he had to use.

Chris was mainly in charge of the PCB. He spent long hours on Protel really early to get the board done, even when other teams had not even finished their schematic. The deadlines we had set for the PCB construction turned out unrealistic, but Chris worked to meet them very closely. When the board arrived, he worked full time once again to get it debugged and running.

However, those three parts are far from making the project complete, and all three of us spent a lot of time working together to integrate the parts. Manolis was doing a great deal of soldering, and debugging on the printed board. Jeremy became an expert of Protel very early and designed part of the PCB. Chris worked both in the low-level routines and the compiler itself, to get the parts integrated.

Looking back, each of us did much more than a third of the project.

### **b. Expectations and Realizations**

Considering the RoboLogo project and team, we realize that we were very lucky. First of all, we were able to achieve our goals, in successfully creating an easy way of controlling interactive robots. Second, even though our separate parts were clearly enough defined to allow individual progress, we all had exposure to all layers of the project, and gained valuable experience.

We managed to finish the project on time, because we set early deadlines, dove right into work, and continued working steadily and effectively. This allowed us to continue making progress and be on schedule, despite drawbacks and unplanned delays. For example, we had not anticipated how long and laborious a process it is to get a PCB laid out, to select the right chip descriptions in our PCB and adapt them to our actual chips, to find the right power regulators, and to correct inaccuracies on the actual printed circuit board.

RoboLogo is a project with many parts that are closely interleaved. This allowed progress on different fronts, without direct dependencies of a part of the project on the completion of another. At all times, all three of us were busy. Manolis was able to write low-level routines and test them on the breadboarded kit, and Jeremy was working on the compiler long before the PCB had arrived. There was always something there to do, and the team was very strong in supporting each other's part, being able to step in and give a hand when help, or simply an opinion was needed. This allowed us each to gain experience about many different disciplines, and have a complete overview of the project.

The RoboLogo project itself exhibited this quality of joining many different disciplines, as a project where custom hardware is best fit in getting things done, low-level assembly routines are suited to the task, but that also integrates a strong software component.

All three of us agree, that this was probably the best team we've had so far. We got along really well, and worked very intensively together.

### **c. The future of iLogo**

We are very happy with the development of the iLogo compiler. Although we did not have enough time to cover all the possible control structures and primitives, we did manage to get quite a few translated programs tested with our truck. After the initial setback of not being able to use the Berkeley Logo parser, we were pleased to find how easy and fast it was to develop our own robust parser using the JavaCC/JJTree tools. The object-oriented design of the compiler is also very pleasing to us. Each stage of the compiler is designed using the Visitor pattern described in the book *Design Pattern* by Eric Gamma, et al. This pattern allows tree traversers to be created as seperated objects, instead of doing all traversals as methods of the nodes of the tree. This keeps the nodes of the tree from gaining a lot of code and allows for state to be saved during the traversal without passing the information along to the separate nodes. It also keeps the

parser fairly unblemished from the original and allows for easy extensions later.

The iLogo language itself seemed a great success. The `DO.UNLESS` construct seemed very well suited. On one side, it fits well with the rest of the Berkeley LOGO specification, and on the other side, it was expressive enough to be a sufficient extension in making a language interactive. This construct in conjunction with the `DO.WHILE` construct allow for a wide variety of situations to be handled, in fact seem to be universal in expressing interrupts and handling routines. We hope that either us or some other team will have some time in the future to put some more work into the project, for it to possibly have an educational use, and a reach outside the scope of this class. We are thinking of talking to the children's museum and the computer museum, to see if they would be interested in having a similar project to demo for children. If teachers spend a little time with the children, before taking them to the museum for an interactive demo, the tool can be of great help in teaching them both about robots and about programming.

## Conclusion

The RoboLogo system is an easy and enjoyable way for children to learn about programming interactive robots. The iLOGO language is a simple but effective language for manipulating the programmable truck. The heart of the iLOGO language, the `DO.UNLESS` statement, is a new structure that doesn't have an concise equivalent in low-level languages such as C and assembly. Hence developing custom tools, our own language and our own compiler has been justified.

By abstracting away from the direct assembly control of interrupt handlers, RoboLogo makes programming interactive robots not only more accessible to children, but also allows experienced programmers to write more complicated behaviors, while keeping the code cleaner. We hope to see more languages exhibiting similar constructs for handling events and allowing interactivity.

## *About this document ...*

This document was generated using the LaTeX2HTML translator Version 0.6.4 (Tues Aug 30 1994) Copyright © 1993, 1994, Nikos Drakos, Computer Based Learning Unit, University of Leeds.

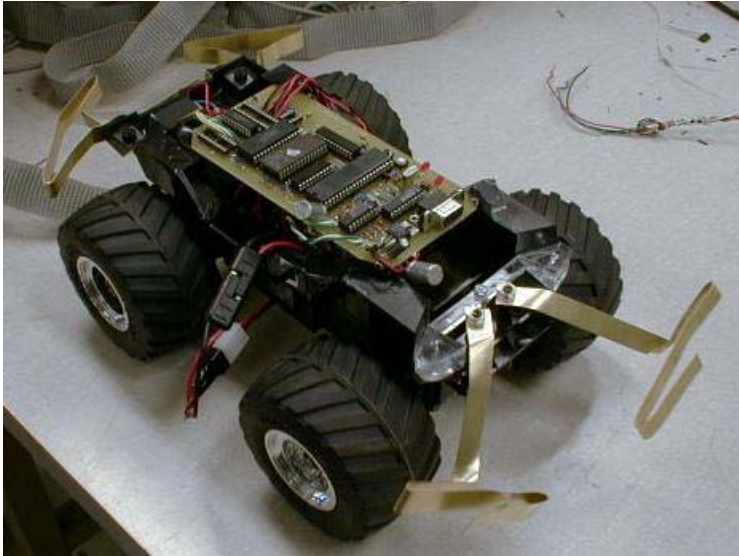
The command line arguments were:

```
latex2html -t RoboLogo -dir /afs/athena.mit.edu/user/m/a/manoli/www/robologo/latex2html/robologo/  
/afs/athena.mit.edu/user/m/a/manoli/www/robologo/latex2html/main.tex
```

---

*Manolis Kamvysselis - Jeremy Lueck - Christopher Rohrs*  
*[robologo@mit.edu](mailto:robologo@mit.edu) Thu Dec 17 1998*

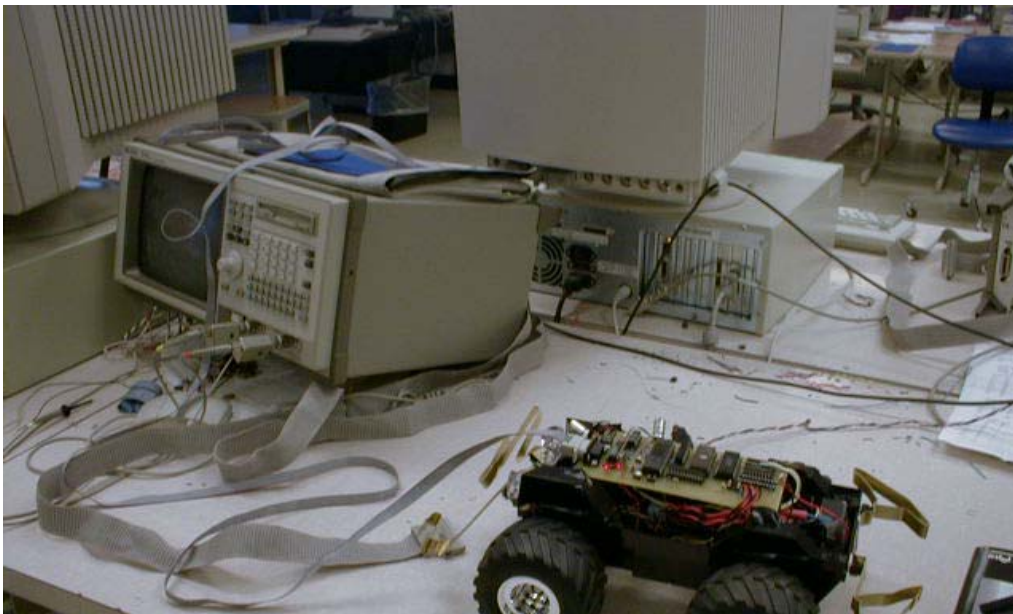
## Robologo Pictures



A close-up of the truck.



The truck attacks Chris!





The truck's birthplace.

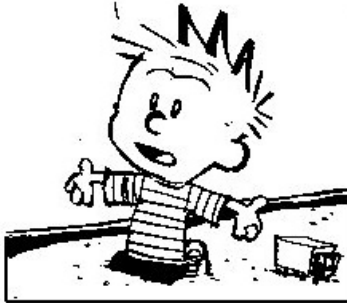
---

Last modified: Tue Dec 29 21:39:01 EST 1998

# Mood - A music recognizer

Auditory recognition and classification based on changes in attentional state.

YOU CAN'T JUST TURN ON  
CREATIVITY LIKE A FAUCET.  
YOU HAVE TO BE IN THE  
RIGHT MOOD.



[Click to see entire strip...](#)

## The team

**M**[anolis Kalvin-selis](mailto:manoli@mit.edu) - [manoli@mit.edu](mailto:manoli@mit.edu)

**O**[vidiu Marina](mailto:strider@mit.edu) - [strider@mit.edu](mailto:strider@mit.edu)

**H**[Obbes-man Vassef](mailto:hvassef@mit.edu) - [hvassef@mit.edu](mailto:hvassef@mit.edu)

**D**[edric Carter](mailto:dedric@mit.edu) - [dedric@mit.edu](mailto:dedric@mit.edu)

## Introduction: What does Calvin really mean?

What Calvin describes here is the main idea motivating our project. To understand a pattern, you have to be in the right mood, the right attentional state. We constructed a pattern recognition system that modelled the state transitions of the human brain, as described by Seymour Ullman and implemented by Sajit Rao for the visual cortex. We used the same states that Sajit used for vision, namely:

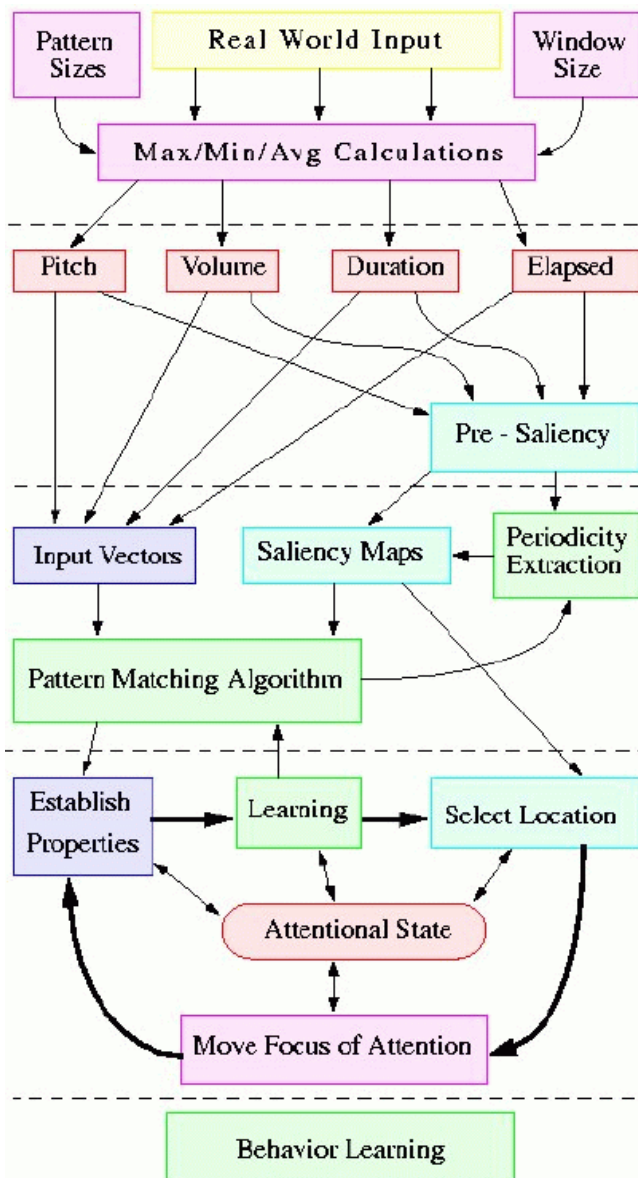
- Select Location
- Focus attention on new location
- Establish Properties at the new attentional state
- Learn low-level patterns and patterns of state changes.

What makes the system able to generalize from one music piece to an entire category is the one thing that remains constant within a class of music compositions: namely, the pattern of changes in attentional state.

---

## Mood - Architecture Overview





### Input Layer

The input provided by the MIDI file is preprocessed to calculate Pitch, Volume, and Duration values, as well as a voice index. The voice index need is replaced by the time elapsed value, which makes the use of voices general enough to allow for a dynamic reordering of voices. Maximum, minimum, and average values within a window are calculated for each of them in a pre-processing step. [Input Details...](#)

### Saliency Layer

Based on these values and their maximums, minimums and averages, saliency values are calculated for each piece of data and their derivatives. This results in saliency maps which make different vectors stand out, depending on what feature one is paying attention to. [Saliency Details...](#)

### Pattern Matching

The pattern matching algorithm will be biased in matching the values which are salient on the vectors which are salient. Patterns of varying lengths will be stored for salient vectors, and matched as new inputs come in. [Pattern Details...](#)

### Prediction

When a pattern is matched partially with the current input, the system will predict future notes as the continuation of the stored pattern. Since more than one patterns will be matched (at least one for each pattern length - one for every pattern table), a weighted average will be calculated, depending on certainty and pattern length. [Prediction Details...](#)

## 1. Vision

### Input-Output Processing is at the heart of Human Intelligence

Some of the research in Artificial Intelligence research has been oriented mainly towards building a general computational theory of intelligence, of which human intelligence would be only one instance. However, some of the recent research in Artificial Intelligence has been primarily directed towards understanding human intelligence, taking the results of neuroscience and psychology into large consideration.

Our research fits in the second category, as a part of the effort to understand general intelligence through a study of animal and particularly human intelligence, the only form of intelligence we know. We use methods of Computer Science, and especially Computer Engineering, as tools to exploit the facts revealed in biology, psychology and neuroscience. Most importantly, **we believe that perception and interaction with the real world is at the heart of intelligence.**

In this perspective, our research is aimed at expressing, with computational tools, a model of human perceptual information processing. In particular, this project is a proposal to explore an **attention-based model for auditory processing**, inspired by the model proposed by Rao [1] and Ullman [2] for visual spatial perception.

### Why Audition, Why Attention?

A lot of work has been done in fields related to visual information processing, for example computer vision, character recognition, etc. However, less work has been done in auditory information processing, in areas other than language processing. Furthermore, only a small fraction of the work on vision is directly aimed at understanding practically how the human brain does it. Similarly, little work has been done in understanding how we listen to and react to sounds other than conversations, for instance, music. How do we understand, interpret, recognize music? Moreover, why does it affect our emotions, our mood?

For the human visual system, Ullman [1] proposed a theory of cognitive routines to account for the flexibility and polyvalence of our visual system in performing various complex information extraction tasks. Rao [2] proposed a model on how the routines can be performed and learned using a generic architecture around attention.

Our task in this project was to try out a model similar to Rao's model for musical perception. We chose music/audition instead of vision, to see if the generic model for spatial information processing is also a generic model for other senses, as Rao speculates. The concept of an attention-based model is an attractive one, for two reasons, which are related to each other:

1. It is what makes Rao's model generic for various visual tasks. It reduces their complexity by organizing them into serial routines, sequences of attentional states. It would make the model generic to all senses if we can apply it here.
2. Attention is at the center of the mystery of the human mind. It is goes together with the unresolved issue of how a machine could have consciousness, and be self-aware. Here, our engineer's approach is the following: if we include manifestations of attention as a computational tool in our model, maybe that will give us insights on the consciousness issue.

## Our Goals, for this project and beyond

- **Understand the human auditory system**
- **Unify the workings of the brain**  
: explain audition and vision under the same architecture, understand the hierarchy of the brain
- **Extend Rao's language of attention to auditory perception**
- **Extend the notion of implicit memory (routines) to understand explicit memory**
- **Attempt to recreate music in a learned style**

## Our Focus for this project

- **Perception of music**  
: to narrow down the extend of this first project, we abstracted out the lower-level auditory mechanisms, and made reasonable assumptions about them. In vision, the evidence from neuroscience and psychology shows that the brain separates the task in two: object recognition and spatial information processing. It allowed Rao to do a correct abstraction of object recognition while he focused on spatial information processing. Similarly, we abstract out the sound recognition process (which among other things, lets us recognize a particular instrument, or spoken language), and focus on the perception of music. To this end we made a set of reasonable assumptions: that the lower-level mechanisms yield to our system a notion of pitch, volume and duration.
- **Learning**  
: we will focus on learning musical routines. Understanding music composition will be left for later work.

## Our Contributions

- **Auditory World in terms of Attentional Patterns**  
: we proposed a new understanding of the auditory world in terms of changes in attentional patterns. Applying Rao's work to audio was no easy task, since at first sight, the two worlds have nothing in common.
- **Periodicity is Learned, not hard coded**  
: New theory of how periodicity emerges from matching successively larger patterns, and understanding emergent patterns.
- **Language of Attention**  
: Provided a language of attention for the understanding of musical pieces. Adding a new category to Rao's system, and filing in the auditory equivalents of Rao's visual routines.
- **Unifying visual and auditory world**  
: Same architecture of attention can be used to explain both visual and auditory worlds. Implications on evolution of those behaviors and on the separation and specialization of our brains to the different tasks it accomplishes today.

## 2. MIDI

### Real World Input

Although we abstract away the issues of note harmonics and playing instrument types, we use a keyboard as input. This allows for a more rapid creation of new tunes, as well as for future real-time input. The same mechanisms used in loading the MIDI file output from the



electric keyboard can also be used in loading pre-made MIDI files.

## Keyboard

Dedric playing puts in the mood. ;o)

The input is limited to one instrument at a time, although multiple notes can be played simultaneously. The notes can be loaded by our utilities from either the MIDI0 or MIDI1 file format. The notes are separated after loading into our system into different voices.

## Time ON / OFF to Durations

Matching notes can be tricky, because the MIDI format stores only time ON/OFF, with a code format which can be confusing. Dedric's program successfully decodes this information and returns a Pitch/Volume/Duration/Time on encoding for each of the notes played. This encoding includes a cleaning up of slight note overlaps.

## Distinguishing Voices

Ovidiu's code, after updating the internal MOOD note store, takes the new note and tags it with a voice. The voices denote notes which overlap, or notes which follow a pattern which is distinct from a separate set of notes being played in the same tune. This is akin to two objects which move about in space: each has a trajectory which is separate from the other object's trajectory. When doing vision, one of the first tasks is to separate potential objects based on their motion. The equivalent is being attempted here in the auditory realm.

As Manolis observes, this is yet another great MOOD success.

On to [Augmenting the Input...](#)

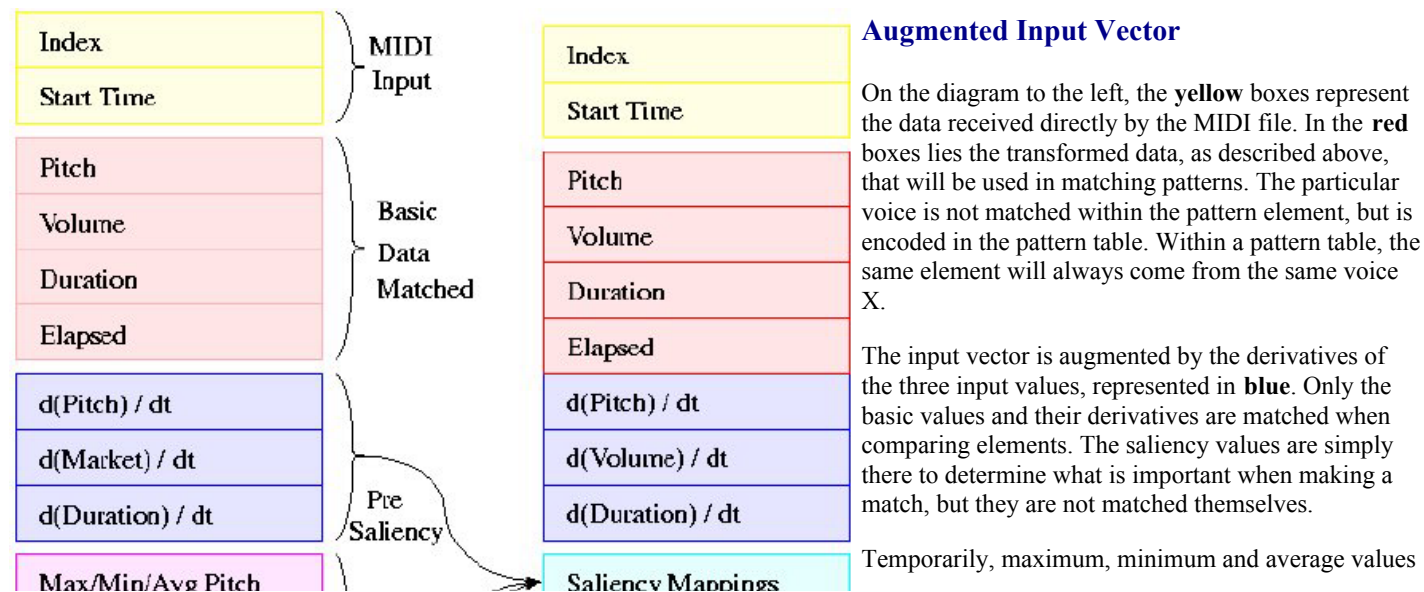
# 3. Input

## Duration Elapsed

The duration elapsed is the time difference between the current time and the time of the note. When a pattern of 5 consecutive notes is constructed, the duration fields will be [4, 3, 2, 1, 0], and they will match exactly any other pattern of consecutive notes. However, there is no constraint on which data will enter a pattern.

For example, a pattern table can have the first element come from voice X, the second from voice Y, and the third from voice Z. A pattern in that table could be "voice X falls 2 notes ago, voice Y rises 10 notes ago, and voice Z has maximum pitch now". In that case the duration fields will be [2, 10, 1] for voices [X Y Z]. The pattern will also match [2, 8, 1], where Y rose 8 notes ago instead. This allows the system to match a larger class of patterns independent of the exact time of occurrence, and which voice the notes belong to.

If such a duration field is not included, one has to train the system to recognize only cases where a pattern depends on exact time relationships. This varying duration field allows for more dynamic relationships between voices. Since pattern tables are created dynamically, depending on what data is salient, the relationships to look for need not be hard-coded. A different pattern table will be created for different combinations of notes, but that's another story, which belongs to the [pattern matching](#) domain.



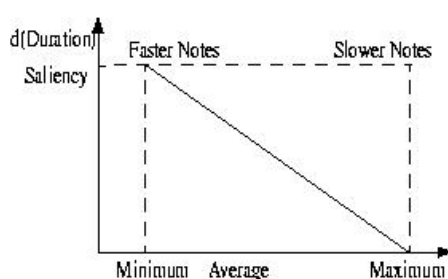
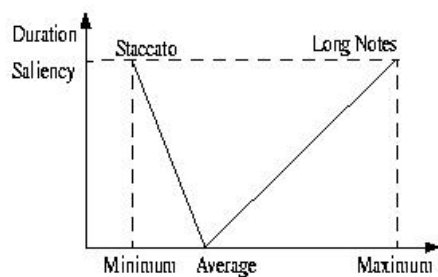
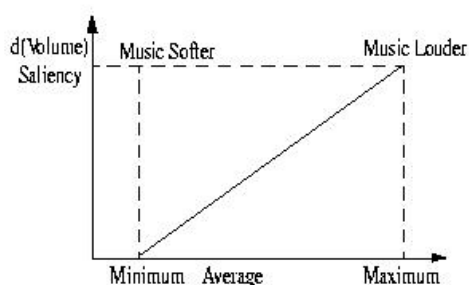
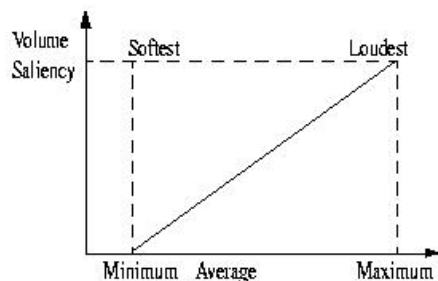
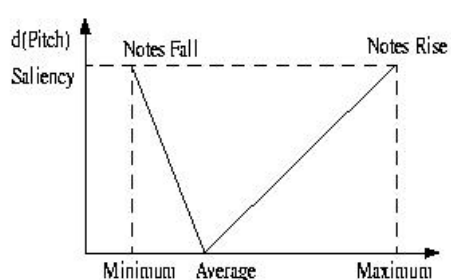
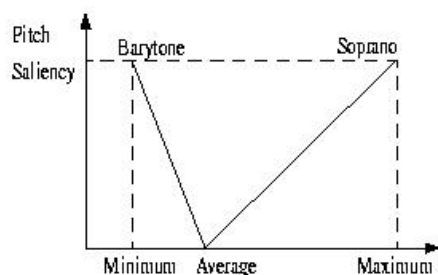
of each of the six fields are calculated for the window-size (**magenta**), and they are used to calculate the **saliency** maps (**cyan**). The window size represents how far back in the past we should look for normalizing values when comparatively analyzing the current data. It is relative to those min max and avg values that values are **matched** within patterns, as will be described later on. They also serve as a normalization step, so that data given in different units can be compared in an unbiased way.

Along with the other saliencies, a periodicity saliency is calculated, which makes salient notes based on a periodically salient history (**green**).

Thus we have reached from a 3-value input (pitch, volume, time) a 7-value vector (pitch, volume, duration, their derivatives, and time elapsed) to match upon. Adding periodicity, each vector also has an 8-field saliency, which determines what values should be matched more carefully.

On to [Saliency](#) or back to [overview](#).

## 4. Saliency Maps



### Vector Saliency

Saliency is a value between 0 and 1 that indicates how important a vector or a field inside a vector is. The most important ones have a value, the least important, a saliency of 0.

The saliency value for a vector is computed as the average of individual saliencies calculated for each of the components.

### Individual Saliencies

Individual saliencies for each value are computed relative to the window of attention. The highest value in the window will generally have a saliency value of 1, as well as the lowest one. An average value will have a saliency value of 0.

The highest volume in the current window yields a saliency of 1. The lowest has a saliency of 0.

The longest duration and the shortest one have saliencies of 1 (either long = salient or staccato = salient), and the average duration will have a saliency of 0.

Similarly for pitch - highest and lowest notes have saliency of 1, and average pitch has saliency of 0.

Derivatives follow the saliency rules as the components they come from.

### Pre-Saliency

Based on the average of these saliencies, each note gets a *pre-saliency* value, which is used to compute the periodicity of the input. This can lead the computer to expect salient notes at specific times if it has been seeing them periodically. Hence, if falling on a periodically salient time, even a silence could appear important.

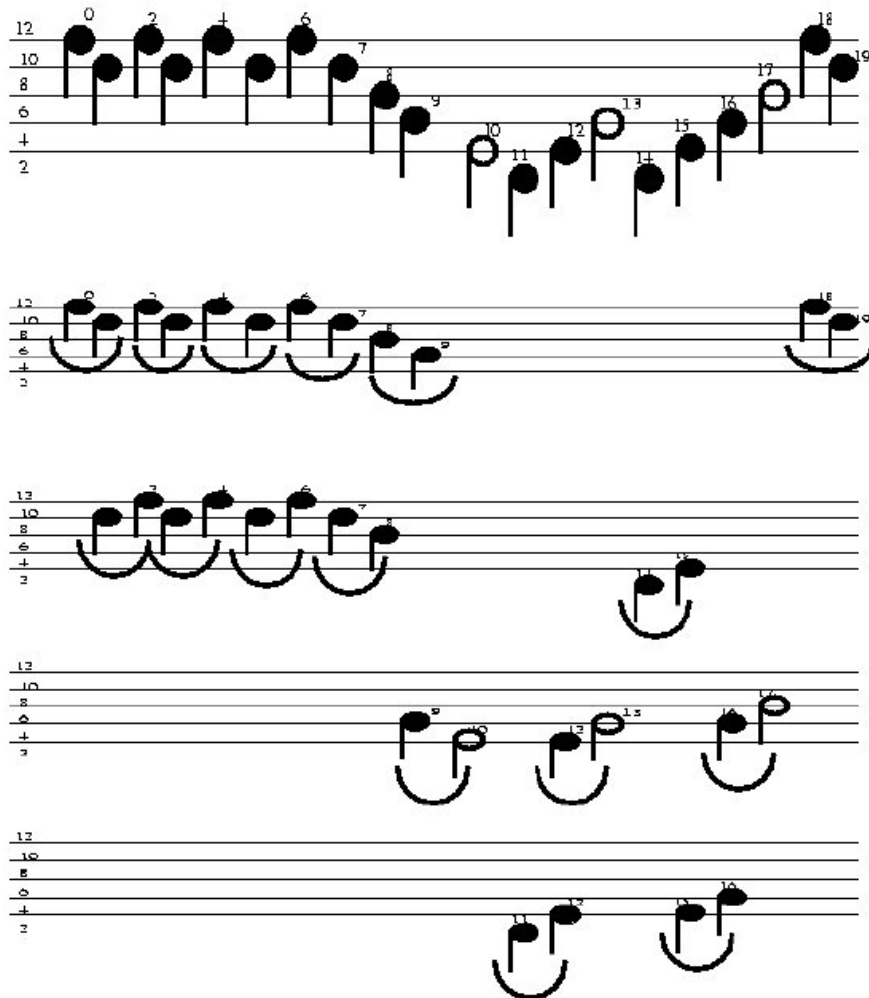
### Saliency Maps

We have thus constructed many saliency maps. One can pay attention to notes salient in volume, or in pitch, or in duration, or in their derivatives. The pattern matcher also computes a periodicity saliency map, which gives a value of 1 to notes expected to be salient, and a 0 to the non-salient ones. Together with the pre-saliency values, this final addition constitutes the saliency part of the code.

On to [Pattern Matching](#)...

## 5. Pattern Matching


### Learning Patterns of Size 2



### Pattern Matching

Pattern matching, along with learning the periodicity of the input, does the basic computations the code is based upon. It is able to learn patterns of different length from the input, and match partially completed patterns to either predict future inputs, or turn the focus of attention to different patterns interesting to a specific behavior.

The pattern matcher receives commands from the attention loop, which directs it on what to match, what to record (learn), and what to pay attention to.

	Pattern 0	Value	Certainty	Value	Certainty
	Pitch	1	1		
	Volume	1	1		
	Duration	2	1		
	$d(\text{Pitch}) / dt$	2	1		

$d(\text{Volume}) / dt$	1	1		
$d(\text{Duration}) / dt$	1	1		



Pattern 1	Value	Certainty	Value	Certainty
Pitch	1	1	1	1
Volume	1	1	1	1
Duration	2	1	2	1
$d(\text{Pitch}) / dt$	3	1	3	1
$d(\text{Volume}) / dt$	4	1	4	1
$d(\text{Duration}) / dt$	4	1	4	1

### Calculating Certainties

Every time a pattern is matched, individual elements get averaged, and their certainties updated. The certainty of a match between two values is one minus the difference between the two values over the difference between the maximum value and the minimum value at the current window. The certainty of a match between two pattern elements is the average of the certainties of their values. The certainty of a match between two patterns, is the average of the certainties of their elements, weighted by the saliency of each element. Intuitively, one will try to match within a sequence of notes the important notes, and will care less about less salient notes which do not match.

---

Table Size 2												
Pitch	1	1	1	1	1	1	1	1	1	1	1	1
Volume	1	1	1	1	1	1	1	1	1	1	1	1
Duration	2	2	1	1	2	2	1	1	2	2	1	1
d(Pitch) / dt	2	2	1	1	2	2	1	1	2	2	1	1
d(Volume) / dt	1	1	1	1	1	1	1	1	1	1	1	1
d(Duration) / dt	1	1	1	1	1	1	1	1	1	1	1	1

Table of Size 3												
Pitch	1	1	1	1	1	1	1	1	1	1	1	1
Volume	1	1	1	1	1	1	1	1	1	1	1	1
Duration	2	2	2	2	2	2	2	2	2	2	2	2
d(Pitch) / dt	2	2	2	2	2	2	2	2	2	2	2	2
d(Volume) / dt	1	1	1	1	1	1	1	1	1	1	1	1
d(Duration) / dt	1	1	1	1	1	1	1	1	1	1	1	1

Table of Size 4																
Pitch	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Volume	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Duration	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
d(Pitch) / dt	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
d(Volume) / dt	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
d(Duration) / dt	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

## Pattern Tables

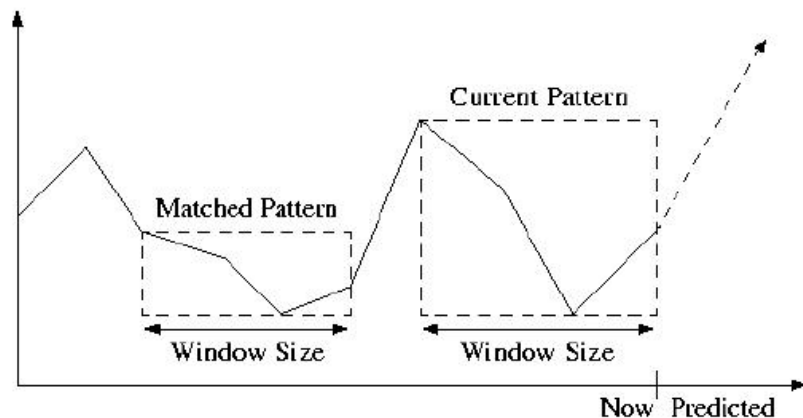
The figure shows an individual pattern of two elements. A pattern table will hold many such patterns. A different pattern table is created for every pattern length that we are matching. Moreover, for the same length (the same number of notes that we match upon), there will be different pattern tables for different voice combinations. For patterns of 3 notes for example, we can have a pattern table storing all patterns found on voices [a b c] another one storing all patterns found on [a b d], and so on.

## Updating Values

When the certainty with which two patterns are matched is too low, then a new pattern is added to the pattern table.

When the certainty is high enough, then the matched pattern is extended to include the current pattern with which it matched.

## 6. Making a match



### Making a Match

In the figure, we would like that the second pattern enclosed in the box matches the first one, previously recorded. Clearly, the values do not match, nor do their derivatives, since everything seems to have doubled (for example in volume).

Therefore, when a match is made, what is compared is not absolute values, but instead values relative to the current maximum and minimum of the window-size considered.

### Making a Prediction

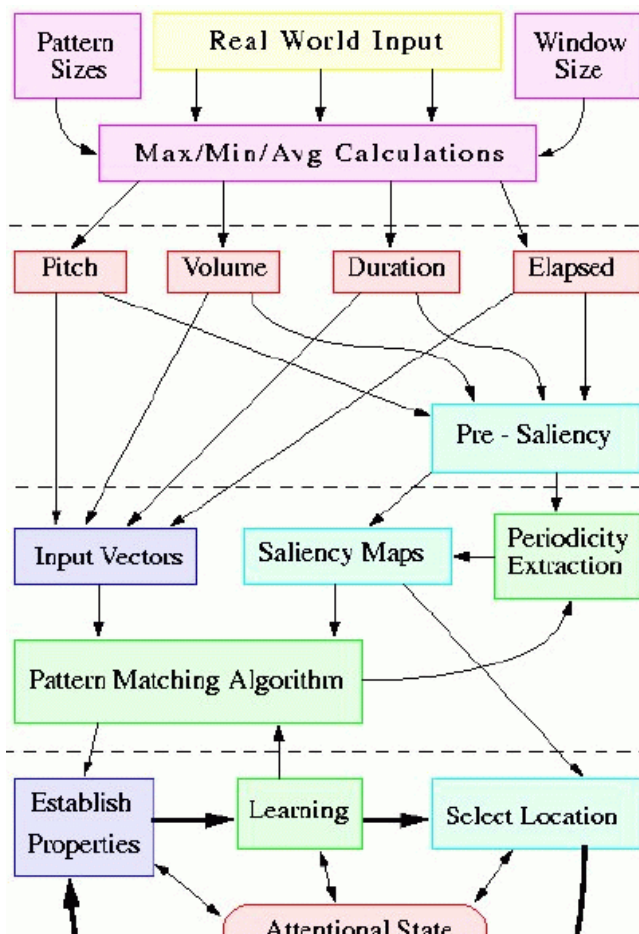
Predictions are made accordingly, relative to the current window maximum and minimum. If the current pattern has length 4, then the system will match the first 4 elements of a stored pattern of length 5, and predict the next note as the final element of the matched stored pattern.

Patterns of different sizes are matched in making the predictions. Their predictions are weighted then averaged, to obtain the final prediction. For each length, let  $\text{best}[\text{length}]$  be the pattern of that length that best matched the current pattern (of length-1). Each  $\text{best}[\text{length}]$  will have a certainty value associated with it, and with each element. This certainty value, along with the pattern length, are used in weighting the different size predictions.

The longer the length of the pattern is, the higher its weight should be. The reason for this bias of trusting longer patterns is that they are less likely to match. For almost anything, a two-note pattern will match with high certainty.

On to [Attention...](#) Mood - Report

## 7. Attentional Loop



The attentional loop is composed of four types of procedures, called serially always in the same order, establishing properties at specific locations, learning patterns, selecting locations, and moving to new locations. A behavior is specified by the commands that are going to be called from each family.

### Establish Properties

This is the largest of the procedure categories. It works on the specified focus of attention to establish local properties.

#### Establish relations between properties

#### High duration followed by rise in volume then silence/applauding

This could be hard coded in our brain. When we hear sssssssssSSSSSSSHHH-BOOM!

#### Determine movement type at current location

A movement could be either sad, or angry, or romantic - determine based on low-level + patterns

### Learning

The procedures in this category are responsible for the learning that gets done in the system. Different types of learning are possible, and they're basically controlling the pattern matching code into doing useful things.

#### Learn patterns of length i-j ending at current note

Records the patterns either learning them as new entries in the pattern table or changing old patterns to be able to match the new input.



**Do not learn anything.**

If the input is not salient, or not interesting, or simply that we're sure that we're already matching something well known, we can simply not record anything.

**Learn pattern of patterns.**

One can simply abstract at a higher level and learn patterns of patterns instead of patterns of notes. Useful for more complex behaviors.

**Select Location**

There are different ways of selecting a location, and different criteria to base this selection upon. A location is either a note or a window size, or a voice to listen to.

**Select most salient overall**

One can simply pay attention to the note that is the most salient in the current window, or to the voice that's most salient

**Select most salient on a particular feature**

The feature can be volume, duration, pitch or their derivatives

**Select based on pattern**

From the pattern matching code, we can select the location that matches optimally the current pattern, or a particular pattern we are interested in

**Select on periodicity**

Location either where the next salient note should appear, or where a periodic pattern should start appear again.

**Move Focus of Attention**

As mentioned above, there are more than one type of locations to move the focus of attention to, and there are more than one ways to move the focus of attention there.

**Move to selected voice**

Changes the current voice, where patterns are recorded from

**Move to selected time**

Changes the current time to be something different than the next time

**Move to selected window size**

Changes the window size based on which saliencies are calculated

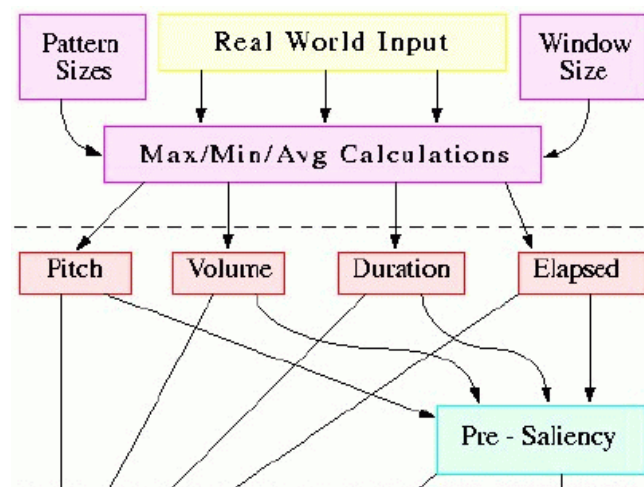
**Move to selected pattern**

Changes the current pattern to which we're matching

**Move completely or partially**

Finally, one can move there completely or still pay attention to many other things

## 8. Behavior



Behaviors are a specification of which procedures are called within each type in the attentional state loop.

**Attentional State**

The attentional state is the shared state which coordinates the different routines of the different categories. They are the means of communication among them. The properties established at the focus of attention will be saved there. The learning routines will keep there the patterns that the pattern matching algorithm matches upon and modifies. The location to move to (along with the type of location it is) will be saved for the MFOA procedure to read it.

It's basically a database of shared state, and provides no functionality by itself, but simply a place to scribble on at every iteration of the attentional loop.

## Choosing Procedures

The choice of procedures is thus crucial in understanding the music that is presented to the system. This choice can be either hard coded, to recognize specific types of music, according to how we think we understand a piece we listen to. A behavior will simply be a series of routines alternating types.

## Behavior Example

To recognize a fugue, the system will

- EP: Number of voices playing
- LN: nothing (no input long enough yet)
- SL: Select location of most salient notes overall
- MF: Move FOA to that voice
  
- EP: Pitch - Duration - Volume - Period to expect for next note
- LN: Learn patterns at selected pitch/duration/volume/period
- SL: Select voice where pattern repeats
- MF: move to voice (or adjust window-size = period when not found)
  
- EP: Changes between theme and its repetition
- LN: Learn changes as new patterns of length = length\_of(theme)
- SL: Select voice of pattern
- MF: move to voice

## Learning Procedures

Alternatively, those procedure sequencies can be learned, by selecting the most salient at each time, and seeing which one it was. When in the future it sees that the first routines emerging by following movements and most salient properties, it will match patterns of changes in attentional state, and follow those emerging patterns as behaviors.

Our hope is that similar behaviors correspond to similar types of music, and thus our program would be able to recognize and categorize types of music simply by the way it reacts to each of them.

# 9. Contributions

## Auditory World in terms of Attentional Patterns

We proposed a new understanding of the auditory world in terms of changes in attentional patterns. Applying Rao's work to audio was no easy task, since at first sight, the two worlds have nothing in common.

## Periodicity is Learned, not hard coded

New theory of how periodicity emerges from matching successively larger patterns, and understanding emergent patterns.

## Language of Attention

Provided a language of attention for the understanding of musical pieces. Adding a new category to Rao's system, and filing in the auditory equivalents of Rao's visual routines.

## Unifying visual and auditory world

Same architecture of attention can be used to explain both visual and auditory worlds. Implications on evolution of those behaviors and on the separation and specialization of our brains to the different tasks it accomplishes today.

# 10. Future Work

## Use more Complex and Accurate Musical Primitives

We reduced the input to a 3-dimensional vector for pitch, duration and volume. This is an oversimplification of our perception. One direction for future work is to explore the lower-level sound wave perception with the correct level of complexity. We would understand



how the many harmonics of an instrument enriches our perception of music.

## **Extend the model to the whole spectrum of Auditory Perception**

Our project expressed a language of attention for audition, within the limits of music perception, given one instrument. With a more accurate model for lower-level auditory perception, we can extend this language to all of the auditory world. We could understand how we perceive several instruments coherently together, or how we can select one instrument, one conversation among others, or one pattern of sounds in the middle of nature.

## **Understand Memory**

Visual and Auditory Routines are nothing else but elements of our implicit memory. With this insight, we can explore how implicit memory relates to explicit memory and understand how it all works in the brain.

## **Levels of Attention - Distributed Attention**

When we first learn how to play an instrument, we must pay very close attention to the movement of our fingers. But once the learning is done, we can play the piece like a routine, without even paying attention to what our fingers are doing, and focus on something else, for example a conversation with a friend. How does attention move horizontally between senses? How does attention move vertically between levels of processing, when it can follow every movement of the fingers or follow the global flow of a routine? Furthermore, how do we distribute attention, in other words pay attention to several things at the same time?

## **Understand the Hardware**

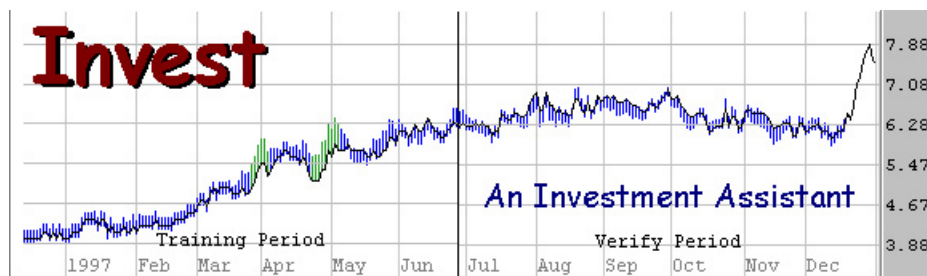
Having a unified model for auditory and visual perception can help us understand the functions of the neural structure of the human brain. How is attention implemented in our brain?

## **Attention and Consciousness**

If we can put attention at the center of all our perception, will that explain the notion of consciousness and self-awareness?

## **Music and Emotions**

What is beautiful about music? How can music affect our mood, make us sad, happy, angry, energetic, or even empowered or enraged? Music has the potential to create a strong emotional response in people, sometimes pushing it to an extreme. For example, Muslims give to some forms of music the same forbidden status as alcohol or gambling, because such music is believed to alienate people. If our model shows how we perceive music, then by studying what patterns of music perception affect our emotional state, we might get insights about how emotions work.



## Invest - An Investment Assistant

6.038 Final Project - AI in Practice

[Manolis Kamvysselis](#) - [Shishir Mehrotra](#) - [Nimrod Warshawsky](#)  
[manoli@mit.edu](mailto:manoli@mit.edu) - [shishir@mit.edu](mailto:shishir@mit.edu) - [nimrodw@mit.edu](mailto:nimrodw@mit.edu)

## Abstract

This paper describes our attempt at implementing a stock price prediction system implementing various AI techniques. We found relatively poor performance across the gamut of techniques attempted. We conclude that stock prices are a random walk across the general mean of the system. Therefore, we propose that it is virtually impossible to predict with a high degree of certainty how a stock price will perform in the future.

## Introduction

The of the project is to build an investment assistant. The purpose is not to have the computer secretly predict the output of a stock, but to use the computational power of computers and the methods learned in Artificial Intelligence in practice, in the domain of the stock market. The system is to help an investor make predictions on the future of a stock, based on the history of the market.

## Definition of the Problem

There are many ways of approaching this problem. Programs have been written to manage entire portfolios. In our case, the user chooses a set of stocks to predict, and a set of stock histories to predict these stocks from. The system is then to make predictions using different AI methods, and judge its competence based on the actual data from the stock market.

More precisely, a program can predict either the future of a single stock, or of a set of stocks. Also, it can predict either an absolute stock price, or the sign of the derivative, that is if the stock will go up or down. Finally, the program can either predict the short-run outputs, or the long-range behavior of the stock.

Each of these alternatives corresponds to a different type of investment strategies. The user might be interested in investing in many stocks if they go up over a long period of time, or he/she may be interested in a few quick investments of stocks that are going to go really high in a short time.

## Structure of the Market

There are different ways of approaching the problem, and they reflect different theories of how the market works. The assumption of the project is that there is a certain correlation between past stock prices and future behavior of the market.

Of course, other factors affect the stock market, such as political decisions, economical trends, hurricanes, international trade, boycotts and strikes. Since we don't have that data, we can either assume that it is reflected in the history of the market (for example that it is the low prices of a company's stocks that caused a strike, or that the economic trends which caused the strike are reflected in the stock market history). Alternatively, we can affirm from the start that our program will not predict accurately such changes in the stock market.

Two different predictors were built on these assumptions. The first one is a Neural Net predictor, that can handle highly non-linear relations between history and future of the stock market. The second one is a Nearest Neighbor predictor, which assumes that linearly similar histories should yield similar results. Finally, a third kind of predictor was designed and implemented, but never tested, trying to pay attention to only those points of the market which seemed important

## The Expectations

What we expect to gain out of the project is not a program that is going to make billions (thus avoiding the case of a program that is going to waste billions). First of all, we're not building a program that will be left alone playing on the stock market. Instead we're building an assistant that will advise Investors on what the correlations are between stocks, and that can throw computational power directed by a human's insights.

We understand that if predicting the market was such a simple task that a simple neural net could predict it, then people wouldn't be trying to make money on it anymore, since everybody would just know the solution. Moreover, in an environment like the stock market, when everyone has found the solution and is making money on the stock market, the solution irremediably changes, becoming how to beat the others who are using that particular solution. It is therefore both in practice and in theory an intractable problem.

Then why attack a problem for which we know there is no solution as a final project for a class? First, because of our interest in the stock market business, and to gain some hands-on experience in building a stock market predictor. More importantly though, we are here to learn, and doing this project we learned a great deal about AI in practice.

---

## Neural Net

In designing the new Neural Network code, it was important to ensure efficiency for feeding the network forward, as well as a relatively efficient way of handling the data structure's elements. The design choice involved building on top of certain of the BPNN neural net code provided by CMU, while completely rewriting and building on top of other parts.

The decision to use the BPNN code stemmed from the fact that it was a relatively robust system with free access to the source code. Furthermore, it had some functionality that was elegantly implemented. BPNN has an incredibly simple way of handling saved nets. However, its handling of weights and feed forwarding were incredibly inefficient. The first optimization involved modifying their squash from being a function, to being an inline define. The second modification involved rewriting their feed forward code. Feed forward is a function that is called an awesome number of times during neural net use. Therefore, the number of function calls that feed forward places are minimized. This leads to tighter code that is quite readable.

Training the net was yet another issue. We knew that the size of these nets could become excessively large, and were concerned that training of the nets using standard backprop would be painfully slow. Therefore a conjugate gradient method was implemented.

## Conjugate Gradient

Conjugate gradient is a minimizing function that relies on the value of a function as well as the value of its gradient. The value of the gradient is calculated as described in Nimrod's individual paper. However conjugate gradient relies on calculation of error over *all* of the training samples. This required a fundamental change in how the BPNN neural net data structure was handled. Because backprop can be trained incrementally as training samples are introduced there is no need to store the entire sample set. However, conjugate gradient does require storing the entire set and changes how samples must be handled. The current implementation stores the entire input set in RAM, thereby allowing for very quick training. The choice of storing the input set in RAM restricts how large the training set is allowed to grow. Though it was feared this would be a lethal constraint, we found that the available resources were more than sufficient to handle the set we were using.

The use of the Numerical Recipes for C code to handle the conjugate gradient algorithm proved to be an excellent choice. Though there was a decent amount of work in designing an interface between the expanded neural net code and the format that the numerical recipes code expected, the transition was relatively seamless requiring only the changing of the types used in the function data structures.

## Encoding of inputs

Inputs to the neural net are open-ended depending on what factors are chosen and how the space is used. The only requirement is that the input values be between 0 and 1. This proved to be a very tricky problem, however, since our scaling caused some numbers to appear inconsequential. We suspect this is one of the major causes of poor convergence in some of the nets. For example some nets show a phenomenally small output error, however when one examines the output, it becomes apparent that the net and the target output are poorly correlated. However, upon further inspection, many of these poor convergences can be explained by examining the values of the output nodes. Since the error is the sum of the squares of the output nodes, a difference of  $10e-3$  actually becomes a difference of  $10e-6$ . It would appear that a difference of  $10e-3$  is inconsequential, but when the price of the feature varies from  $4 \cdot 10e-2$  to  $5 \cdot 10e-2$ , it becomes apparent that this "inconsequential" difference is in fact very large. We recognized some methods of dealing with this problem, however, chose to avoid them for simplicity. Some solutions include changing the nodes in the net to linear functions instead of the sigmoid, or scaling different factors differently.

The arrangement of inputs to the neural net was relatively consistent. To describe the system I will present a simplified net. Assume that we are concerned in predicting IBM's stock tomorrow based on IBM's value today and yesterday, as well as Dell's stock today and yesterday. In our encoding system this leads to 2 factors (IBM and Dell) with each factor having a "history" of 2 (today and yesterday). Therefore, the input of the neural net includes 4 nodes. The output of the neural net is simply IBM's value tomorrow which is encoded in a single output node.

## Problems

Conjugate gradient appears sporadic at times in what it learns. We were receiving output values either pegged at 1.0 or at 0.0. Some of these can be explained by poor encodings of the inputs, but we have not been able to explain some of the other poor results.

## Results

The neural network code was mostly disappointing. Though convergence appeared to happen very quickly, the net itself did not perform well. On tests where we would look back at predictions and see how a broker would have done with the network, we were getting very poor results. For example, one of our tests involved looking at the predicted data and finding when it had a maximum and a minimum. We examined how a portfolio would have performed had the user sold on the max day (at the real value for that day) and bought on the min day (at the real value for that day). Our net performed poorly against the optimal performance and sometimes even lost money.

Changing of the net size did not have a magnificent effect on output. We settled on a neural net that had a hidden level of 16 hidden nodes.

---

# Nearest Neighbors

## The concept

The reason to try Nearest Neighbors is that there is no need for training. The concept is simple: For every output, the prediction of the next value is simply the output of that feature vector whose input is the closest to the input values. In that case, closest means that the sum of differences between the two feature vectors is minimal.

The advantage of a Nearest Neighbor approach to prediction is that the user can change portfolios and immediately get a predictive result, without wasting time in training. If the result seems promising, then a neural net (or other predictive measure) can be used to verify the "goodness" of a new portfolio.

## Problems

The problem with nearest neighbor is that to minimize the error it's better off by predicting the middle value at all times. Will be less likely to pick up small changes.

---

# Pattern Matching

## Characteristics of the Stock Market

The pattern matching alternative to Neural Nets and Nearest Neighbors was built based on the following assumptions on the Stock Market Characteristics:

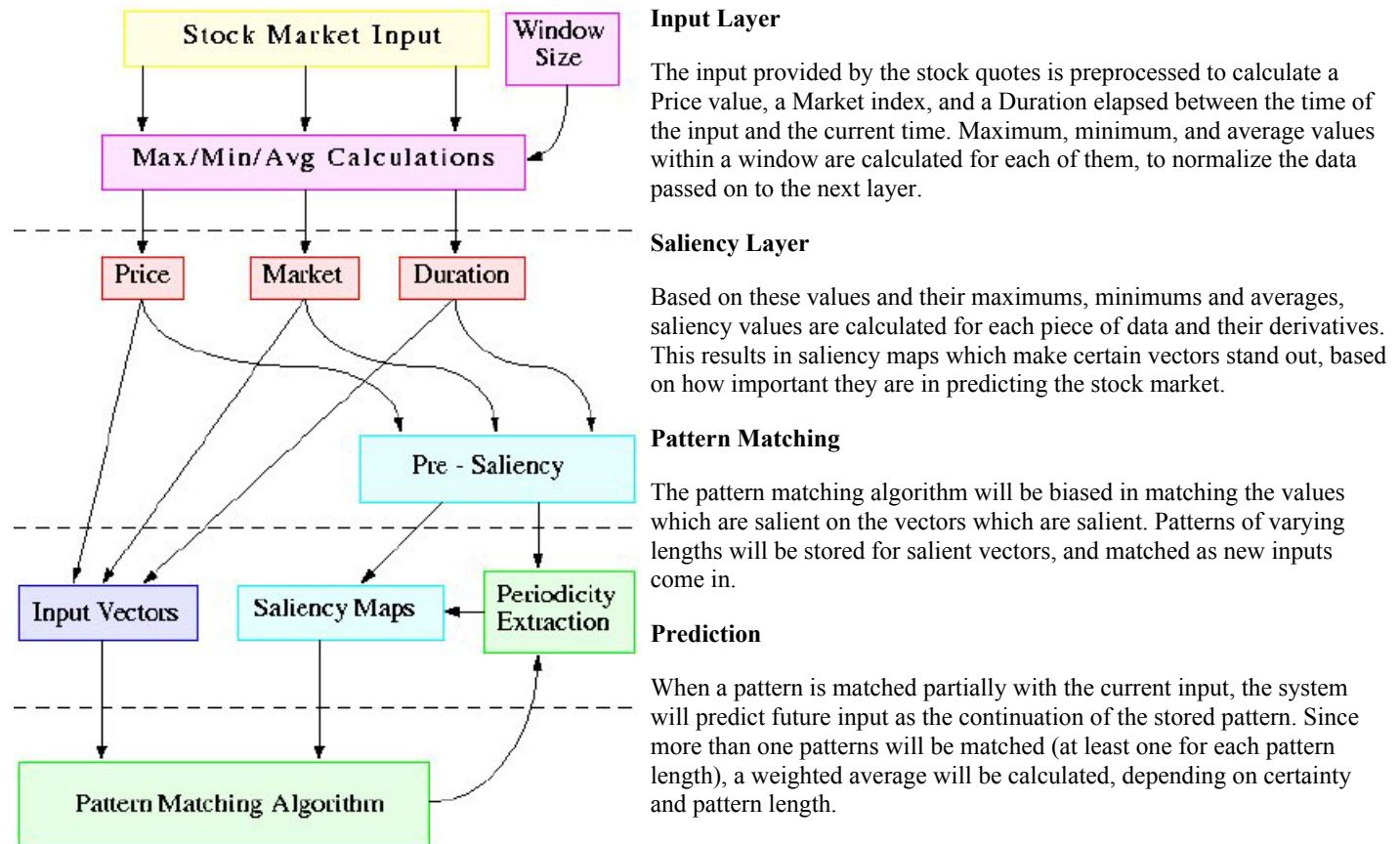
- The important features of the stock market are hidden by too much information
- Everyday fluctuations are erratic and unpredictable
- Patterns can be recognized in the structure of the fluctuations of the stock market
- An uncertain guess can be dangerous if not labelled as less certain
- A stock's outcome depends on factors largely spread in time and between stocks
- Additional data, such as value derivatives or market averages might be easier to predict and to predict from

## Features and Design Criteria in the Pattern Matching code

To address those problems, the current design is as follows:

- The system's goal is to predict important changes, and not everyday fluctuations
- Certainty values are added to the predictions
- A measure of saliency was incorporated that assigns an importance value to each data
- Patterns are matched only on salient data so as to ignore everyday noise
- Relations among stocks are not hard-coded but dynamically extracted as they arise
- The training time is summarized in one pass on the data, learning and predicting at each step
- A market index and stock derivatives are used along the stock values in matching patterns

## 1. Architecture Overview



## 2. Input

### Market Value and Duration Elapsed

The market and duration fields supplement the stock value, by providing information otherwise not contained in the succession of values. The market index represents the state of the market at the time the stock was quoted.

The duration elapsed is the time difference between the current time and the time of the stock. When a pattern of 5 consecutive stocks is constructed, the duration fields will be [4, 3, 2, 1, 0], and they will match exactly any other pattern of consecutive stocks. However, there is no constraint on which data will enter a pattern.

For example, a pattern table can have the first element come from stock X, the second from stock Y, and the third from stock Z. A pattern in that table could be "stock X falls 2 days ago, stock Y rises 10 days ago, and stock Z has maximum value now". In that case the duration fields will be [2, 10, 1] for stocks [x y z]. The pattern will also match [2, 8, 1], where Y rose 8 days ago instead. This allows the system to match a larger class of patterns independent of the exact time of occurrence, and what type of stocks are involved.

If such a duration field is not included, one has to train the system to recognize only cases where a stock depends on exact time relationships. This varying duration field allows for more dynamic relationships between stocks. Since pattern tables are created dynamically, depending on what data is salient, the relationships to look for need not be hard-coded. A different pattern table will be created for different combinations of stocks, but that's another story, which belongs to the pattern matching domain.



### Augmented Input Vector

On the diagram to the left, the **yellow** boxes represent

the data received by an individual stock quote. In the **red** boxes lies the transformed data, as described above, that will be used in matching patterns. The stock value becomes the price (after some normalization). The time is encoded in the duration elapsed. A general value of the stock market at the time of the stock quote is encoded in the Market Value. The stock name is not matched within the pattern element, but is encoded in the pattern table. Within a pattern table, the same element will always come from the same stock X.

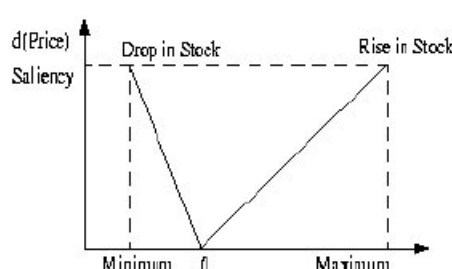
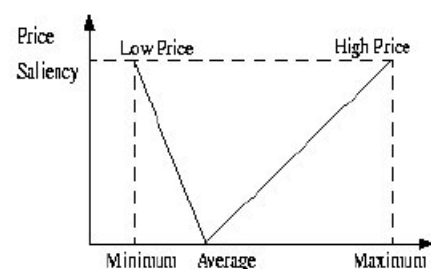
The input vector is augmented by the derivatives of the three basic values, represented in **blue**. Only the basic values and their derivatives are matched when comparing elements. The saliency values are simply there to determine what is important when making a match, but they are not matched themselves.

Temporarily, maximum, minimum and average values of each of the six fields are calculated for the window-size (magenta), and they are used to calculate the saliency maps (**cyan**). The window size represents how far back in the past we should look for normalizing values when comparatively analyzing the current data. It is relative to those min max and avg values that values are matched within patterns, as will be described later on. They also serve as a normalization step, so that data given in different units can be compared in an unbiased way.

Along with the other saliencies, a periodicity saliency is calculated, which makes salient vectors based on a periodically salient history (**green**).

Thus we have reached from a 3-value input (time, name, value) a 6-value vector (value, market, duration and derivatives) to match upon. Each vector also has a 7-field saliency, which determines what values should be matched more carefully.

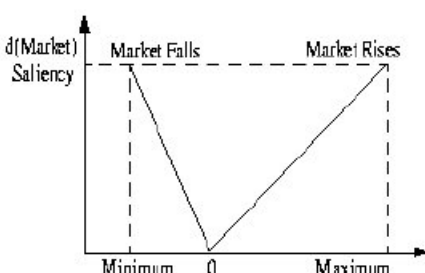
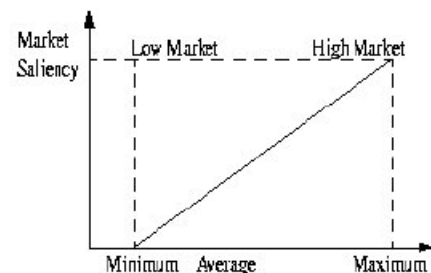
### 3. Saliency Maps



#### Vector Saliency

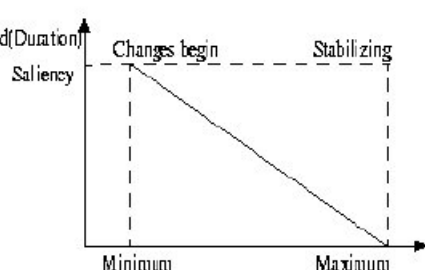
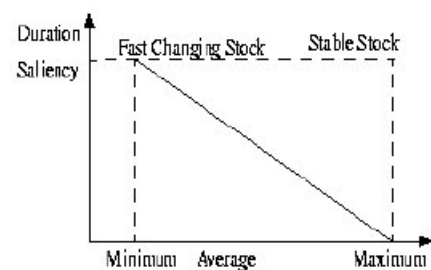
Saliency is a value between 0 and 1 that indicates how important a field inside a vector is. The most important ones have a value of 1, the least important, a saliency of 0.

The saliency value for a vector is computed as the average of individual saliencies calculated for each of the components. A vector saliency is also between 0 and 1.



#### Individual Saliencies

Individual saliencies for each value are computed relative to the window of attention. The highest value in the window will generally have a saliency value of 1, as well as the lowest one. An average value will have a saliency value of 0. The distribution was made linear in between for reasons of simplicity.



For example, one should pay attention to a stock when it's at its maximum value. Similarly important are the moments when the stock is at its minimum value. However, when the stock is just an average value, no special attention should be paid to that particular point in time.

Similarly, a maximally rising stock will be salient (highest stock value derivative), just as a maximally falling stock. A stable stock will have

a derivative of 0, and a saliency of 0 accordingly.

The market saliency functions in the same way more or less, only the system is now biased to paying attention to high-markets. The duration saliency is biased to those stocks that change faster, as well as those which start changing faster.

These biases can be changed for best performance as the system gets tested. In doing so, insights might be gained about real market behaviors and relationships. The first-order function that is now used in saliencies could also be changed to second order, or gaussian as the system becomes more mature.

#### Periodicity Saliency



Based on the average of these saliencies, each stock gets a *pre-saliency* value, which is used to compute the periodicity of the input. This can lead the computer to expect salient stocks at specific times if it has been seeing them periodically. Hence, if falling on a periodically salient time, even a non-salient stock could appear important and be recorded in a pattern.

Pattern matching, along with learning the periodicity of the input, does the basic computations the code is based upon. It is able to learn patterns of different length from the input, and match partially completed patterns to either predict future inputs, or turn the focus of attention to different patterns interesting to a specific behavior.

The pattern matcher receives commands from the attention loop, which directs it on what to match, what to record (learn), and what to pay attention to.

### Saliency Maps

We have thus constructed many saliency maps which can make different vectors stand out, depending on what feature one is paying attention to. One can pay attention to stocks salient in price, or its derivative, or in a high market environment, or a rapidly changing market, or a rapidly changing stock, or a stock which starts changing. The pattern matcher also computes a periodicity saliency map, which gives a value of 1 to stocks expected to be salient, and a 0 to the non-salient ones. Together with the pre-saliency values, this final addition completes the saliency maps constructed on the input.

## 4. Pattern Matching

Pattern 0	Value	Certainty	Value	Certainty
Price	.4	.7	.38	.9
Market	.6	.6	.59	.6
Duration	2	.2	3	.2
$d(\text{Price}) / dt$	-.01	.9	-.02	.85
$d(\text{Market}) / dt$	.01	.4	-.01	.5
$d(\text{Duration}) / dt$	1	.8	1	.8

### Pattern Tables

The figure shows an individual pattern of two elements. A pattern table will hold many such patterns. A different pattern table is created for every pattern length that we are matching. Moreover, for the same length (the same number of elements that we match upon), there will be different pattern tables for different stock combinations. For patterns of 3 elements for example, we can have a pattern table storing all patterns found on [x y z] another one storing all patterns found on [a b z], and so on.

### Pattern Matching

Pattern matching, along with learning the periodicity of the input, does the basic computations the code is based upon. It is able to learn patterns of different length from the input, across different stocks. A match partially completed can predict future inputs, or turn the focus of attention to different patterns interesting to specific markets behavior.

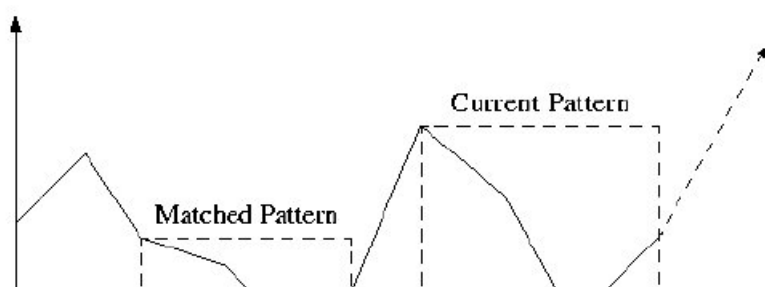
### Calculating Certainties

Every time a pattern is matched, individual elements get averaged, (weighted by the number of inputs each value has matched), and their certainties are updated. The certainty of a match between two values is one minus the ratio of differences between the two values and the maximum and minimum values at the current window. The certainty of a match between two pattern elements is the average of the certainties of their values. The certainty of a match between two patterns, is the average of the certainties of their elements, weighted by the saliency of each element. Intuitively, one will try to match within a sequence of stock the important features, and will care less about less salient features which do not match.

### Updating Values

When the certainty with which two patterns are matched is too low, then a new pattern is added to the pattern table. When the certainty is high enough, then the matched pattern is extended to include the current pattern with which it matched.

## 5. Predictions



### Normalizing for a Match

Let's consider a simple example where all elements of a pattern are consecutive stock values of the same stock.

In the figure, we would like that the second pattern enclosed in the box matches the first one, previously recorded. Clearly, the values do not match, nor do their derivatives, since everything seems to have doubled (for

example in volume).

Therefore, when a match is made, what is compared is not absolute values, but instead values relative to the current maximum and minimum of the window-size considered.

### **Making a Prediction**

Predictions are made accordingly, relative to the current window maximum and minimum. If the current pattern has length 4, then the system will match the first 4 elements of a stored pattern of length 5, and predict the next stock as the final element of the matched stored pattern.

Patterns of different sizes are matched in making the predictions. Their predictions are weighted then averaged, to obtain the final prediction. For each length, let `best[length]` be the pattern of that length that best matched the current pattern (of length-1). Each `best[length]` will have a certainty value associated with it, and with each element. This certainty value, along with the pattern length, are used in weighting the different size predictions.

The longer the length of the pattern is, the higher its weight should be. The reason for this bias of trusting longer patterns is that they are less likely to match. For almost anything, a two-stock pattern will match with high certainty.

### **Conclusions**

The system was never tested on actual stock market input. It was tested on hand-written patterns, and it successfully recognized them and matched them. If such patterns exist in the stock market was thus something we were not able to assert.

A major concern i have for the system's success is about the progressively decreasing certainty values for each of the patterns as more and more vectors are matched. The solution that was envisioned but not implemented was a reordering of the pattern table after the certainties reach low values. That would take the extreme vectors out of the set of vectors matched with the particular pattern and match the other vectors in the set with one or the other. This would hopefully split the mass of uncertain vectors by splitting its centroid to create smaller and denser sets.

Another problem is that of patterns which start out exactly the same but turn out different. The system will be 100% sure of one prediction when it has matched exactly previous state, and it will be 100% wrong. Solutions to this were tested by increasing the pattern length and decreasing the minimum saliency for considering more inputs, in which a divergence should be found, but such a divergence was not evident. This depended, of course to the specifics of the input fed into the system.

---

## **System Integration**

### **File I/O**

The interface between the middle engine and the neural network code and nearest neighbor code interacted through files. This allowed development to occur concurrently and led to code that was isolated.

Files had a structure as follows: A file includes a header that describes how many features, how many outputs, how long in history the "window" should be opened and how many days total exist in the database. The file is then separated into an "input" set and an "output" set. The rows in the input set correspond to the different features being examined. The columns correspond to different days. The file is read into memory building a table of features against dates. Again, accesses are quite efficient, but introduces some limit onto the input size.

### **Test program**

Finally, a test program was implemented to generate the neural net and nearest neighbor results given the output of the middle engine. This system built ontop of all the other systems as the culminating point through which all relevant functionality was provided.

---

## **Results**

After building all of these systems to model the stock market came another huge obstacle: testing it. The major questions to be answered were:



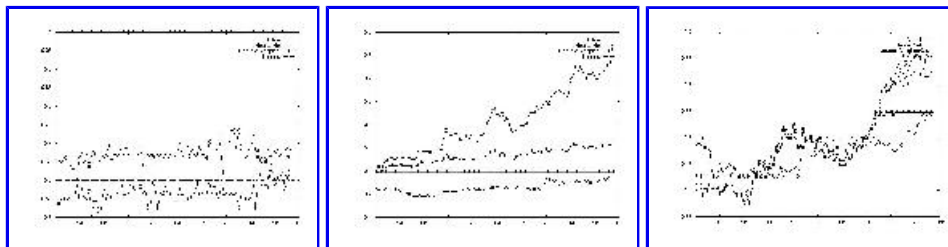
- Was the system working as expected?
- Could the system accurately predict prices in the stock market?
- What information did the system need to make good predictions?
- Could the system accurately predict movements in the stock market?
- Could the system produce buy/sell strategies for the market?

The first step in answering these questions was coming up with data. A large amount of stock prices for the past 6 years was obtained from Sloan School of Management's Virtual Trading room. However, there were a number of problems with this data. It was very large, and in a cumbersome format, but most importantly, all the stocks that it modeled did not have data for the same range of dates. Because of these restrictions, the number of stocks was whittled down until we were left with 148 stocks. Unfortunately, they ended up being a rather random selection of stocks and thus establishing correlations, such as between markets, was very difficult.

The second step was to represent this data in useful ways. One of our original predictions was that modeling stock prices may be difficult but capturing other secondary information may be easier. Thus the concept of a transform was added to the system. These *transforms* took one set of data and manipulated it to produce a piece of data structured the same way. Some examples of transforms that were used are: shift data, minimum value, maximum value, running averages, etc. These transforms also provided an easy mechanism for producing testing data from training data, it just needed to be shifted over in time.

At this point, all of our questions could be answered. The results are summarized below, and for each strategy, three outputs are provided, first one where the system performed poorly, a second one with medium performance, and a third one where the system performed well.

## Test 1: Predict what you learned

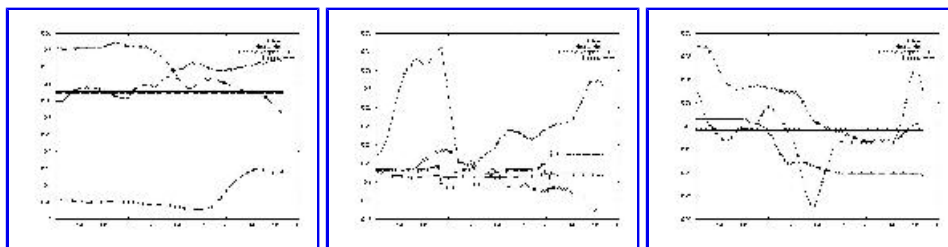


The purpose is to ensure that our system was working in the way we expected. To do this, the inputs to the system were the same as the outputs. All the system had to learn was the identity function.

The neural net performed extremely well. This showed that the Neural Net was definitely capable of modeling such a complex system given enough information.

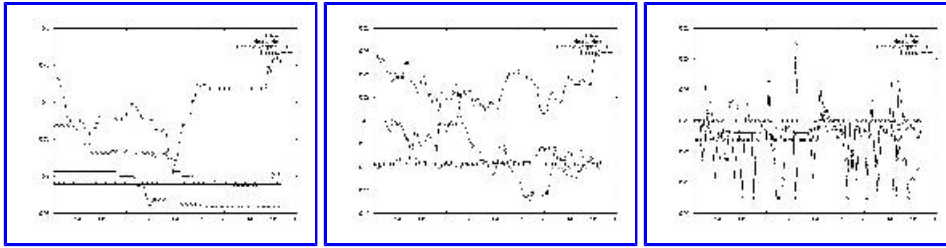
Nearest Neighbors worked very well but it was expected too. Every piece of training data was the same as the testing data, so it had seen each of the vectors already. It just had to look them up. Again, this test showed us that Nearest Neighbor could work, given a large database of old values.

## Test 2: Smooth out stock value



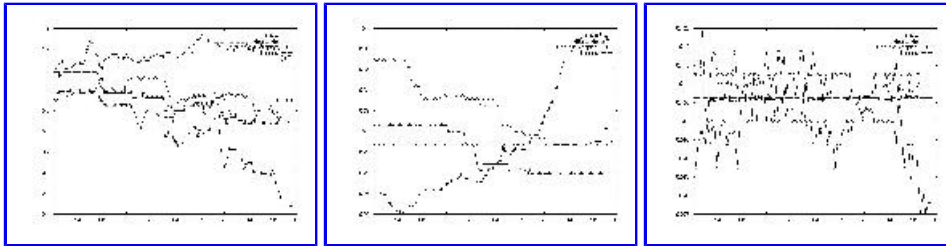
One of the largest problems in predicting stockmarket information is that the data is so erratic. Since the data was sampled on a daily level, this was seen as a huge problem. Thus the stock's value was smoothed out by establishing a running average transform, i.e. computing the average of the next week's data as today's value.

## Test 3: Stock on Its Own History



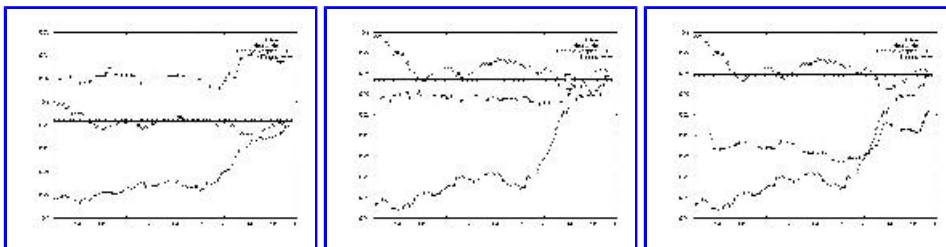
Perhaps one of the most used estimates for a stock's performance is its past performance. Many analysts will say things like, "Last time it went up like this, it came down even harder." Modelling this effect was the subject of this test. A very simple shifting transform was used where day 1's data was replaced by day  $x$ 's data as day 2's data was replaced by day  $x+1$ .

## Test 4: Stock on History + Market



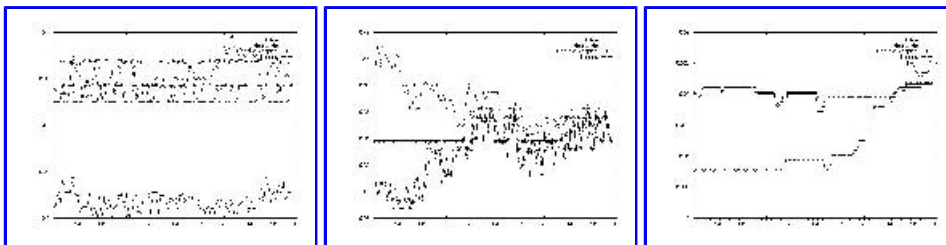
Another indicator of a stock's value which is often used, is the performance of other stocks. The idea is that the stocks act together as a market, and incorporating a market index could be very beneficial. To this end, a sum transform was used which averaged all the data for the 148 stocks.

## Test 5: Market on Stock X



The opposite question from Test 4 was asked in this test: how does this one stock determine the market index's value? By testing each stock's correlation against the average computed in Test 4, this correlation could be established.

## Test 6: Strategy Output



Having the system output predictions for stock prices is very interesting, but in order to use the information one must know have a strategy. In other words, one could look at the system's predictions and determine when to buy and when to sell. Perhaps a very useful implementation of the system would be to have the system itself report profitable buy days and profitable sell days.

The idea of having strategy be an output can be very complicated. Different types of strategies exist, from ones that capitalize on short term fluctuations to ones that lay in wait of long term trends. For the purpose of demonstrating the possibility of having a strategy output, a very simple strategy model was chosen. Basically, the model said that given a set of predictions, one would pick one day to buy and one day to sell. One would prefer to buy on the lowest price day and to sell on the highest. Thus, as two additional binary outputs (1 - buy/sell, 0 -no buy/ no sell), a simple strategy output was attempted.

---

## Conclusions

### Near Misses

(It sure sounds better than failures.)

We were able to predict the stock market (NOT!). Just like we were expecting, the program will not make billions on the stock market. Most of the time it gives a wrong prediction, and sometimes it's really off.

The pattern matching code was never integrated to the system. It worked on hand-written input, but was never tested on actual stock prices.

Our user interface fell in the water as we lost a member of our team.

### Successes

Our first success was getting 10 years worth of data, in a world where everyone is selling stock quotes at their own price, and trying to keep everyone else from getting to the stock market history.

Another success was getting the neural net and nearest neighbor code integrated with the data, and learning the actual market values. The system was able to predict what it was trained on, and thus effectively exhibited learning.

Finally, we learned a great deal about how team projects should be done, and the mistakes ones should avoid. We learned about training Neural Nets and the conjugate gradient method of training. We learned about nearest neighbors and its limitations.

### Market Behavior

The most important lesson we come out of this project with however, is that the stock market is not an easy thing to predict, and that simple AI algorithms cannot effectively extract the important information out of it. Either more complicated algorithms should be devised to predict the stock market, or more data should be accounted for in the prediction.

The short term behavior of the market seems to be more or less random, and one should be interested more in larger patterns, possibly by smoothing out the data to eliminate high-frequency noise.

A simple advise to those betting money on stock market computer systems would simply be "Don't!"

### Future Work

Now that the system is built and manages real data, different engines can be incorporated and tested. Different algorithms might have greater prediction power, perhaps in predicting other types of values (integrals, derivatives, combinations of prices), or based on other types of values (hurricane rate, political movements).

---

## The Source Code

### See handouts

---

## Individual Contributions

### Manolis

- Pattern code

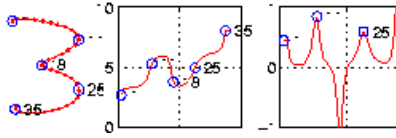
- Nearest code
- Training Testing
- Write-up

## Shishir

- Middle Engine
- Designing Tests
- Training Testing

## Nimrod

- NN code
- Nearest code
- Data acquisition
- Designing Tests
- Training Testing



# Digit Recognition in Curvature Space

Manolis Kamnitsis

## Introduction

### Abstract

This paper presents our experience with an curvature space for online character recognition. The  $(x,y)$  coordinate data along the character curve is transformed to local curvature data. The peaks of this curvature represent the sharp turns made by the pen in constructing the character. The height and timestamp of these peaks are used as features. The number of peaks for a single character rarely exceeds five or six. Thus, we have transformed the 256 dimensional data of off-line recognition on a  $16 \times 16$  grid, to less than 10 dimensional data.

This aggressive reduction in dimensionality is certainly lossy. However this paper makes the claim that the curvature peaks preserve the information most important to character structure. To demonstrate the claim, we construct a classifier for handwritten digits that bases its decisions solely on the few peaks in curvature space.

The results are promising: 200 digits of each character were correctly classified based on a HMM mixture model constructed from 300 examples of each digit. Unsupervised learning techniques are also envisioned. The character data plotted in the lower dimensional space suggests distinct clusters emerge, that could separate and group trained symbols, as well as recognize new symbols as categories.

### Algorithm Overview

The first step of the algorithm is a compression problem, to reduce the dimensionality of the data, while maintaining a meaningful set of features. The  $(x,y)$  character data set is first transformed to  $(dx,dy)$  increments along the path of the pen. These increments are transformed into (angle, distance) increments, and finally the derivative of the angles data is used for the representation. Our measure of the curvature at a point in the character curve will be the value of the angle derivative at that point. A very important denoising step is applied to this derivative data, to filter out unimportant variations and bring out the meaningful variations. The peaks in this denoised derivative show the points of highest curvature. These points correspond to the moments when the pen was changing direction, in other words, the transitions between consecutive sub-strokes within the same character.

These points of highest curvature are used as features. The set of (derivative, index) pairs for each peak are what we base our recognition upon. We train an HMM network to cluster the data from each character. The representation for a character becomes then a mixture of Gaussian centered on the clusters of points in the (derivative, index) space. A new character is classified by comparing the likelihoods obtained by each of the models when explaining the peaks of this new character.

### Organization of the paper

The next part of the paper gives an overview of data preprocessing steps. The third part outlines the transformation to curvature space. The fourth part of the paper explains the noise reduction mechanism and its performance. Section 5 details the feature selection step. Section 6 goes over the training of the HMMs on the data. Section 7 evaluates the performance of the algorithm. Section 8 concludes with applications of this work and possible extensions. All the figures have been provided in appendix, so that they can be made as large as possible without interference with the text. The legends of the figures will be given within the text.

## Working with online data

The online character data was kindly provided by Nick Matsakis (matsakis@ai.mit.edu). It contains 500 examples of each of the 10 digits obtained on a Cross handwriting recognition pad. Nick was preprocessing the data, and we kept his preprocessing code. He resamples the data uniformly, along the curve, to obtain the same number of points for all data samples. This processing step will not be described in this paper.

After uniform resampling, each character in the data set has 36 points, in order, in  $(x,y)$  coordinate form. All characters have the same length, and they are centered at zero. Their orientation is not normalized however, and the same character could be rotated in different

ways.

The  $i$  coordinate is an index from 1 to 36, implicitly provided by the ordering of the points. The problem of online data recognition seems to at least as easy as off-line recognition, since one can always simply drop the  $i$  indices, and fall back onto an off-line recognition problem, where the order of points is not known.

## Relation with Offline Data

The problem is that even though the keystrokes of two people when writing a 4 can be different, their written outputs are usually similar. This is because we humans have tailored our writing to match the expectations of an off-line recognizer, namely other humans. We are taught to write so that the pixel output looks like the desired character, regardless of what series of steps (strokes) we take to achieve it. People often optimize for speed and ease of writing consecutive characters, so the order ends up being often the same across different writers. This work didn't have to cope with this problem, since Nick Matsakis has preprocessed the data to reverse each non-regular stroke. Some irregularities remain, of course, and they will not be solved by simple preprocessing steps. We accept them though as different representations of the same character.

This section compares the two problem domains, and shows how the dimensionality of online data is intrinsically lower. The next section will describe how we further reduce this dimensionality in angle derivative space.

Online data for character recognition is intrinsically easier to deal with than pixel data. We are a step closer to the production of characters, since the information about individual strokes has not been lost on a static paper (or pixel-map). We have all the information about the curves that the pen traverses in space, in order to write the character down, and we can use the structure of this curve to our advantage.

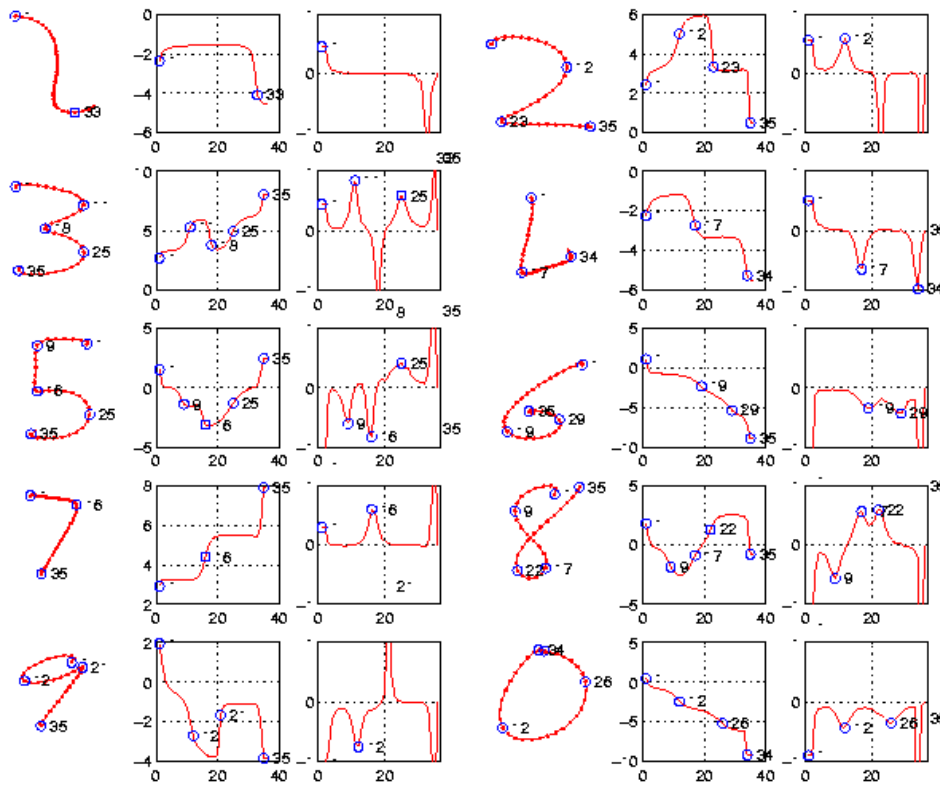
In fact, since characters can be recognized independent of their scale, we can guess that spacing of the different points is not what is important in the recognition process. The only time when scale enters in the recognition process is when multiple characters are placed next to each other and a capital O for example, is distinguished from a lower case o. In fact, if we sample at equidistant points along the curve

## Online data is lower-dimensional

We will take advantage of the fact that online data is available to reduce the dimensionality of our problem. Compare a pixel-based character recognition dimensionality in a 16 by 16 grid. Each pixel can be on or off, a total of 256 dimensions. However, a much smaller number of actual data points was used in creating the character. Principal components analysis on the 256 dimensional data reveals that with only 30 dimensions the data can be represented accurately enough to yield less than 10% classification error. In fact, our online data has an (x,y) coordinate for each of the 36 samples, hence 72 dimensions. If we disregard lengths, and only keep angle data, we have come down to 36 dimensions. This seemed to be the minimum given by PCA. Going further seems pointless, since classification seemed to deteriorate greatly with anywhere less than 10 dimensional data.

## Character Transformation to Angle Derivative Space

### Getting familiar with angle derivative space



We encourage the reader to take a moment now to look at the figures in the appendix. Three types of plots are presented in figure 1. The first column (and the fourth) show the means of the 500 samples of each character in our data set. The characters were averaged per index. That is the coordinates of the third point of digit 5, for example, are the average of all the coordinates of point 3 of every sample of digit 5.

The first thing to notice is how smooth these characters look. If characters looked like that out of the character capture system, then our recognition task would be much easier. We will show how denoising can restore smoothness to original curves in the next section.

The second thing that strikes is that the first character doesn't look anything like a one, and the fourth seems to be missing something. The explanation for digit 4 is simply that nick has software that handles single strokes, so we only get the first stroke of a two-stroke 4, missing the vertical bar that was coming next. As for digit 1, we postulate that this curved shape comes from the fact that it is an average over different orientations of 1.

Now, one can go more into detail in understanding the angle derivative representation intuitively. Take the example of character 3. After an initial discontinuity that we ignore as noise when the pen of the user is put on the table, the derivative goes up, then it decreases sharply, then it goes up again, and finishes with a discontinuity that we ignore again as noise before the pen of the user leaves the table. The increase and then decrease in derivative values corresponds to a right turn along the path of the stroke. The narrow and highly negative value corresponds to a sharp left turn. Then the right turn is repeated for the second loop of the three. The peaks in the derivative and their corresponding turns on the character are circled in purple on the figure. I would recommend going over the different characters and understanding those peaks to get familiar with the representation before proceeding further in the paper.

Figure 2 shows the same peaks that were highlighted by hand in figure 1, picked out automatically by the algorithm in the angle derivative space, and reported onto the angle space, and the character curve. The derivative axes are enforced to go from -1 to 1, to make comparison between derivatives across character easier. This chops off some derivatives that were higher than 1. We allowed those curves to extend past the axes on figure 2b.

## Motivation for an angle derivative space

Now that this angle space is more familiar, let's first see the motivation behind it as a representation for digits. First of all, one needs to point out that the reason why pixel-based classification (and even PCA dimensionality reduction) worked so well with little effort is that the data had already been normalized. First of all, the data was centered on zero, since pixel-based approaches are location specific. Then the data was scaled so that it has total length 1. Finally Nick went through great effort to get the character representation invariant to rotation. He applies a linear transformation (a combination of rotation, scaling, shear) to every character in its curve form in order to align it optimally with the rest of the data, before it was transformed onto pixels. The hard part was to get a transformation that is not character specific, and hence can be used for classification (as opposed to a transformation specific to threes, for example). What makes the  $(x,y)$  coordinate space unfit for recognition is that uniform global modifications, such as rotation, transform the space in a non-uniform way. Different coordinates will change in different ways, which makes recognition hard without a sometimes involved normalization step that

should get rid of irrelevant noise while preserving features important to recognition.

A curvature approach is inherently translation invariant, scaling invariant, and rotation invariant. Under any of these transformations of the input set, the representation will be the same. The angles taken in this representation are not global angles of every point with respect to say the center of the character, but instead local angles on the coordinate derivatives from one point to the next. When a character is rotated, the angles all shift together, since they are based upon increments to the angle of the first segment of the character. Hence, global transformations of the character do not transform these local angles. We argue therefore that an angle derivative space is best fit to a character recognition problem, being invariant to changes in the input that are irrelevant to recognition, and making recognition relevant features stand out, such as the three curves in the digit 3, along with their sharpness and direction.

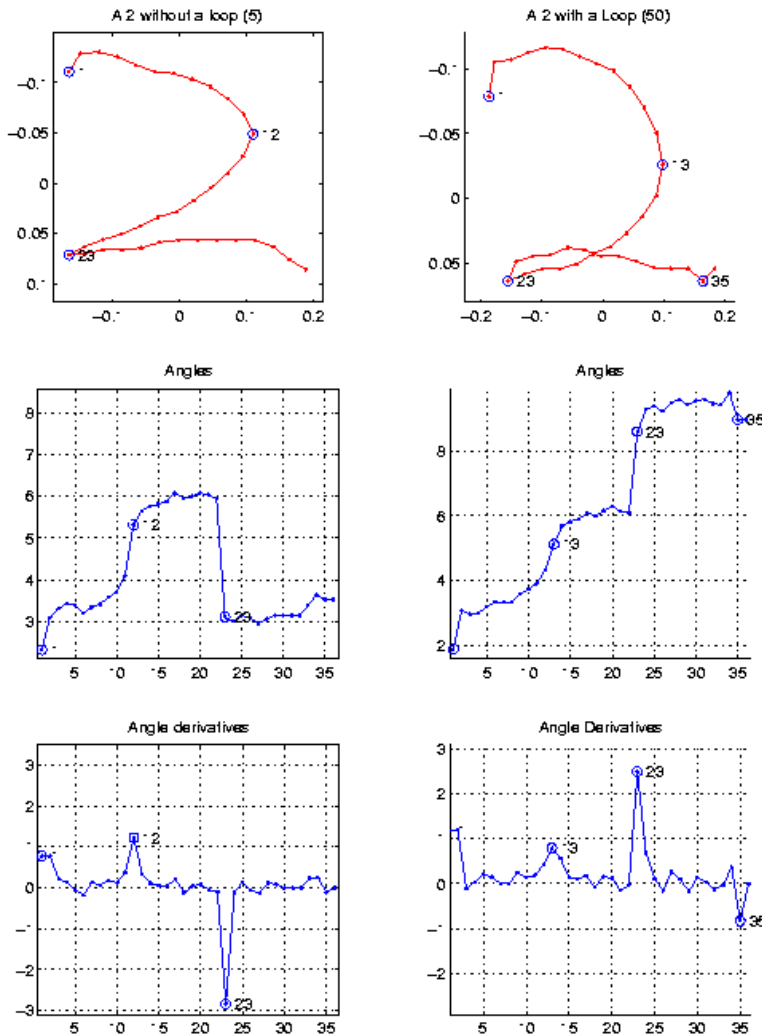
## Resolving angle ambiguities

The main problem of an angle space are the ambiguities due to the periodicity of the data. In a space constrained between 0 and  $2\pi$ , when an angle varies smoothly from 6.2 to 6.3, the angle seems to jump from  $2\pi$  down to nearly 0 causing jumps in the derivative. To get around this problem and obtain smooth data plots, we allow the angles to cross the boundaries, but force them to preserve smoothness in transitions. That is when we have to choose which equivalent representation to give to the next angle down the line, we will choose the one that is most similar to the current one. In other words we add or subtract a multiple of  $2\pi$  to  $\alpha_{i+1}$ , to enforce

$$|\alpha_{i+1} - \alpha_i| \leq \pi \quad (1)$$

Angle derivatives have a range of  $\pi$ , whereas angles have a range of  $2\pi$ . Intuitively, think of moving around the unit circle. If you ever have to take a step great than  $\pi$  in one direction to reach a particular point on the unit circle, you can simply move in the opposite direction, and the distance to travel will be now less than  $\pi$ , since the two ways of getting there must sum up to  $2\pi$ .

When this smoothness constraint is applied to the angle derivatives, the corresponding angle plots can span ranges greater than  $2\pi$ .



An example of such a plot can be found in figure 4. When digit 2 is drawn with a loop, the curvature along the curve never changes sign.



In logo terms, this 2 could be drawn by a turtle that keeps turning right. After the turtle turns 360 degrees, it is facing back in the original direction, but continues to turn, which yields a span greater than  $2\pi$ .

We seem to have cured therefore what seems to be a great problem with angle plots, that two curves could be very similar, and yet their angle plots could look very different because of the periodicity of angle space. Now up to a constant shift along the y axis, two curves of similar characters will look the same.

## The loop ambiguity

Figure 4 illustrates another ambiguity that we need to account for. In a sharp corner, such as the bottom left corner of a 2, or the middle curve of a 3, writers will draw a loop instead of a sharp corner. These loops are sometimes important to recognition. For example, imagine keeping only the bottom part of the twos in figure 4 (after indices 12 and 13). The bottom of the left digit can be interpreted as a less than character (<), whereas the bottom of the right digit can be interpreted as the greek letter alpha ( $\alpha$ ). This shows that our eyes are not indifferent to a loop when we see one, and the two kinds of twos are different characters, both representations both as the number 2. In fact when i came to the US, I personally had a hard time recognizing American 2s with overemphasized lower loops. Hence we could simply consider the two characters to be different, and perhaps an unsupervised clustering algorithm would instruct us so. When we get a little further in the paper, we will show how the loops appear in our learning system and how we deal with them.

This loop ambiguity explains an artifact we see in figure 3. Figure 3 shows the average characters, this time averaged over angle derivative space. We see that orientation is sometimes not preserved (in the case of 7, 9, and 6), but that otherwise, the characters look very natural (in the case of 3, 5, 8, 0, and even the tilted 7, 9 and 6). The major concern is the averaged 2 which is missing its bottom loop. The loop has degenerated to an almost straight line (some indication of an angle shows at the position indicated by the arrow, but there is clearly something wrong). What goes wrong is the fact that we are averaging angles in a euclidian space, as opposed to a periodic space more natural to angles. Hence, the two types of twos shown in figure 4, having derivatives similar in magnitude but opposite in sign, will cancel each other and leave the averaged angle derivatives of the 2 almost flat at that point in the curve (the second turn of 2 went from being the dominant feature in both cases of figure 4, into being much smaller than the smaller peak in the original character).

In fact, this loop ambiguity is very easily interpreted in the derivative angle space. The values for the derivatives at point 23 seem very different when laid out on the plane, but they are in fact close in angle derivative space. One should think of the angle graphs as wrapping around themselves along the y dimension, as if they were the unwrapped side of a cylinder laid out on the paper. Since our smoothness constraint (equation 1) ensures that derivatives will be constrained within a range of  $2\pi$ , the space wraps at  $\pi$  and  $-\pi$ , and very negative derivative values are close to very positive derivatives. The error function in our learning algorithm should be able to handle this property of periodic variables. Our cost function should incorporate some notion of periodicity of the form

$$\Delta(\alpha_1, \alpha_2) = |\alpha_1 - \alpha_2(\text{mod}2\pi)| \quad (2)$$

which would yield an error function of

$$E(\alpha_1, \alpha_2) = (\alpha_1 - \alpha_2(\text{mod}2\pi))^2 \quad (3)$$

along the angle dimension (to be then combined with errors from other dimensions).

## Working with Periodic Cost functions

We are not the first ones facing the problem of error estimation in an angle space. Solutions have been proposed for dealing with periodic cost functions. Bishop and Legleye (1995) propose an approach based on a mixture of kernel functions, that are themselves periodic. They replace a Gaussian kernel based on the x,y coordinates of speed  $v_x$  and  $v_y$

$$\phi(x, y) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{(v_x - \mu_x)^2}{2\sigma^2} - \frac{(v_y - \mu_y)^2}{2\sigma^2}\right\} \quad (4)$$

with a new distribution of the form

$$\phi(\theta) = \frac{1}{2\pi I_0(\sigma)} \exp\left\{\frac{1}{\sigma^2} \cos(\theta - \theta_0)\right\} \quad (5)$$

known as the *circular normal* or *von Mises* distribution, where  $I_0$  is the zeroth-order modified Bessel function of the first kind.

An alternative is suggested in an exercise in Bishop's book (exercise 6.8 p.249). A density function is constructed for the periodic variable  $\theta$  by replicating the range  $(0, 2\pi)$  an infinite number of times to construct a density in the  $(-\infty, \infty)$  range.

$$p(\theta) = \sum_{L=-\infty}^{\infty} p(\theta + L2\pi) \quad (6)$$

This density allows comparison between similar values *mod* $2\pi$  without extra effort once the new distribution has been constructed. To make this approach practical the summation can be restricted to finite values of  $L$ . This will give a good approximation for Gaussian models with exponentially decaying tails.

In our case however, the difference between small negative curvature and small positive curvature matters. Negative curvatures correspond to left turns and positive curvatures to right turns in the character path. On the contrary near  $\pi$  positive curvatures are very similar to near  $-\pi$  negative curvatures, because of the loop ambiguity. Hence, even though our representation angle derivatives goes from  $-\pi$  to  $\pi$ , we will do comparisons into the space from 0 to  $2\pi$ , with negative angles mapped accordingly. Hence our error function becomes:

$$E(\alpha_1, \alpha_2) = |T(\alpha_1) - T(\alpha_2)|^2 \quad (7)$$

where

$$T(\alpha_i) = \alpha_i(\text{mod}\pi) \quad (8)$$

Moreover, we only consider angles greater than a threshold (.4) since small angles could simply be due to noise in the input. Hence, we will not introduce a lot of near-zero angle derivatives which would occupy the two ends of our new space.

## Enriching the representation

We would like our character transformation to be reversible. That is, we would like to be able to recover the character from the angle derivative representation. The reasons for this are multiple. First, we would like to check that our representation is correct. Looking at the quality of the reconstruction we can obtain, we can judge of the quality of the representation, and have a measure of information loss.

Secondly, we want to process the signal obtained in the angle derivative space, and judge what these changes correspond to in the character domain. Signal processing steps could be compression, noise reduction, smoothing, or resampling.

The derive operation is lossy, losing a constant at every order. Hence we need to enrich the representation with an angle constant, representing the orientation of the original character. Moreover, the position derivative from which we calculate the angles loses the absolute (x,y) position of the original character, and lastly moving to angle space loses the lengths.

To enable reconstruction, we add four constants to our representation.

- The (x,y) coordinates of the centroid of the original character.
- The mean segment length of the original character. If the sampling was uniform, this should be enough to recover the original character. Often it will not be an exact reconstruction, but the difference is very small and often imperceptible in our experiments.
- The orientation of the original character, explained below.

The orientation is encoded as the principle angle of the original character. We could have included any particular angle, to enable moving back from the derivative angle space to the angle space. However, since after signal processing particular angle derivatives can change, relying on a single value for the reconstruction would not have generated a stable representation invariant to noise. Hence we decided to use a value for a representative angle of the original character. This angle is given by the largest eigenvector of the original character. We perform a principle component analysis of the (x,y) coordinates in the original character, and find the direction and angle of maximal variation.

$$\text{principleAngle}(\{(x_i, y_i)\}) = \text{cart2pol}(e_1(\{(x_i, y_i), \forall i\})) \quad (9)$$

where  $e_1(\{(x_i, y_i)\})$  is the eigenvector corresponding to the largest eigenvalue of the covariance matrix of all coordinate pairs in the original character.

Our reconstruction will be constrained to preserve the angle of that principle direction. To enforce that constraint, after reconstruction, we calculate the principle angle of the reconstructed point set, and adjust all the angles by the difference in the angles of the two principle angles of the original character and of the reconstructed character. The angle update is

$$\alpha_i := \alpha_i - \Delta(\text{principleAngle}), \forall i \quad (10)$$

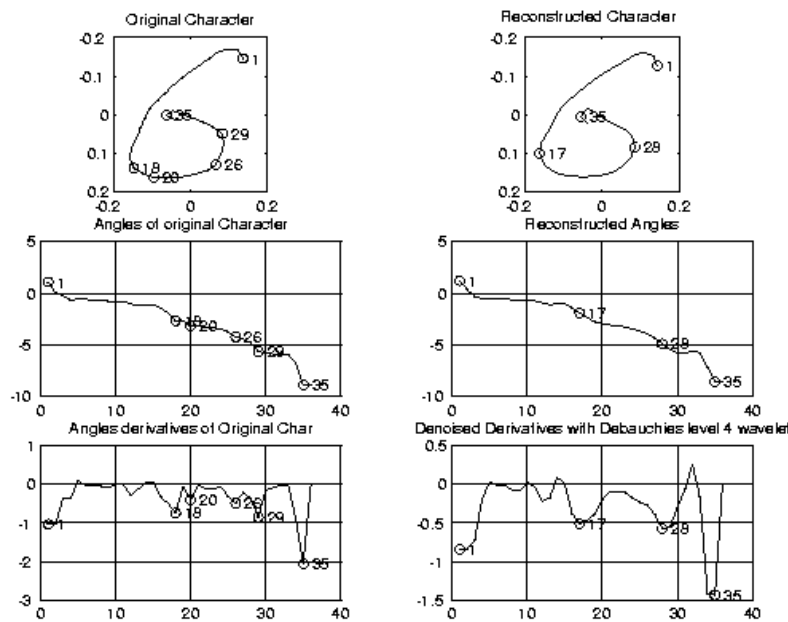
where

$$\Delta(\text{principleAngle}) = \text{principleAngle}(\{(x_i, y_i)\}) - \text{principleAngle}(\{(x'_i, y'_i)\}) \quad (11)$$

where  $(x'_i, y'_i)$  are the reconstructed points.

The results of adding these four constants to our representation worked effectively in making the transformation reversible and the reconstructed characters almost identical to the original ones.

## Noise Reduction



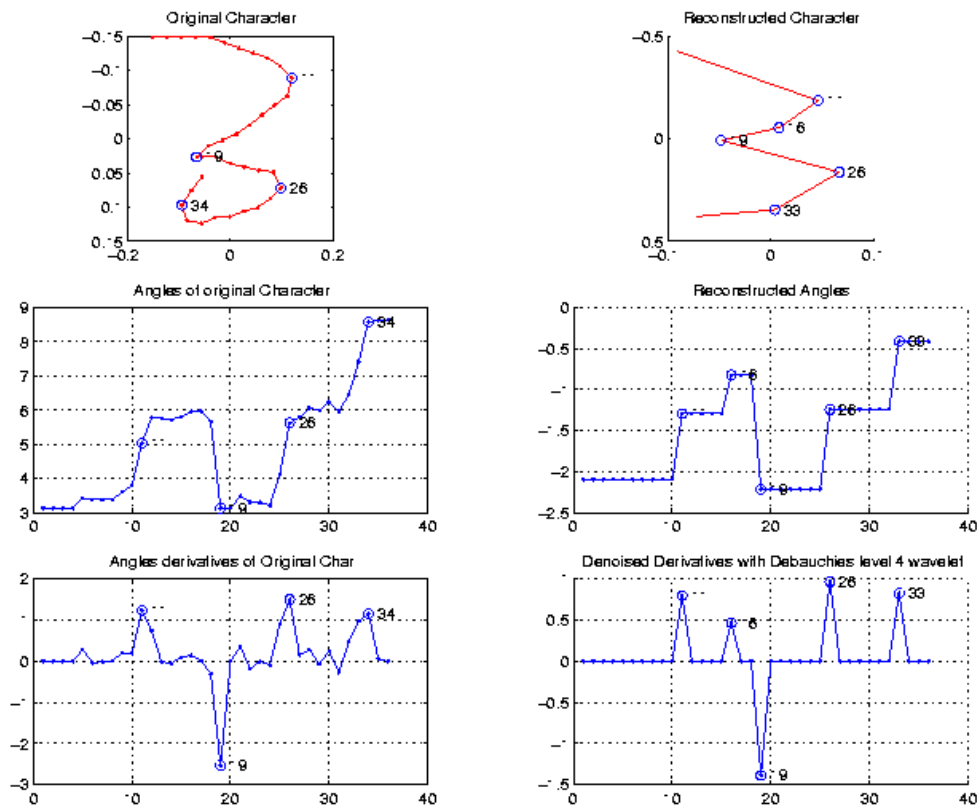
When the entire character set is averaged, the curves obtained for the character, as well as their angles are very smooth. This creates very clean derivatives, with distinct peaks, which let us guess that recognition can be done in the derivative space. However, when a single curve is considered, the signal is often too noisy to find the distinct peaks. We had to find a solution that would filter out the high-frequency variations due to the shaking of the user's hand. To illustrate, we would like to go from a highly noisy signal for a 6 such as the one on the left of figure 5, to a clean signal like the one on the right of figure 5, that resembles much more closely the signal for the average 6 character, shown in figure 1.

## Fourier methods for denoising

A low pass fourier filter for noise reduction did not work, since the highest derivatives are necessary for encoding features in our curvature space. The fourier analysis did not seem to distinguish between the useful high frequencies due to features in the character transcribed from those high variations created when entering characters to the computer.

An experiment to try would be to reconstruct the curvature signal from a mixture of low frequency and high frequency components. We did not have time to perform this experiment.

## Brute force compression for denoising



Another form of denoising would be to completely eliminate the derivatives below a certain threshold. Figure 6 shows an example of such a aggressive compression transformation. The derivatives below 0.4 were simply set to zero. This preserves the most important variations of the curve, at the peaks of curvature, but the smoothness of the curve is lost. There is a low-frequency stability that should be preserved in the character, and it was destroyed in this case. The obtained reconstruction is very coarse (top right of figure 6).

The extra peak at 16 appears from a pre-processing to the aggressive compression step. This preprocessing is in fact the wavelet based denoising filter that we describe next. This step not only added a peak to our reconstruction of the 3, but it also lowered the peaks of the angle derivatives, by smoothing out the signal. The next section provides more details on the denoising.

## Wavelets reduce noise and bring out features in the curve

The hardest part in character recognition is noise reduction. If characters didn't have the meaningless variation due to noise, then we would simply use a nearest neighbor classifier and match the closest labelled curve every time a new perfect character was presented.

We argued that the angle space is useful for recognition but our argument won't hold unless we show that noise reduction is easy in that domain. In fact, using wavelets, the noise reduction step has very nice results, yielding a smoothing of the curves in curvature space, and thus very cleanly marked features with few irregularities.

We use a 4th order daubechies wavelet basis for noise reduction. The properties that make this wavelet basis a good choice is the fact first of all that it is an orthogonal basis with a compact support. We used soft thresholding based on a principle of Stein's unbiased risk threshold selection rule. We experimented with other wavelet bases and error functions, but the best performance was obtained with the combination described.

To judge the performance well the noise reduction works, we compare the curves obtained by noise reducing individual characters with the smooth curves obtained on the average characters. If the curves exhibit the same features, the recognition step will be possible.

In our case, the features are the peaks in the angle derivatives. These peaks occur when the angle derivative is maximal, that is when at the local minima of curvature along the character curve. We circle the peaks and number them with the index of the point at the peak in the derivatives, the angles, and the character (see figure 5).

The original signal had many local minima of curvature that was making it difficult for features to stand out. The denoised version of the signal has clearly marked peaks, that occur precisely where the average character had its peaks (compare with character 6 in figures 1 and 2). The method seems to be robust and the results are promising. Classification performances without this denoising step and with this denoising step are exhibited below.

	without denoising	after denoising
ones	87.96%	94.0541%
twos	70.24%	87.8679%
threes	83.68%	97.8378%
fours	39.4%	53.5736%
fives	97.64%	80.1802%
sixes	83.8%	73.9339%
sevens	3.56%	2.8228%
eights	35.2%	71.4114%
nines	70.12%	86.1261%
zeros	70.48%	82.9429%

## Feature Selection

Once this representation is fixed, and we have successfully reduced the noise in the data, we have to extract features for recognition. Up until now, the transformation was lossless. An exact version of the character could be retrieved from the curvature angle derivative representation, given the appropriate lengths. The noise reduction step is not applied in the construction of the representation, but as a prerequisite to extracting the features, in order to make the features extracted independent of the input noise. Moreover, we showed that a good reconstruction of the original character can be obtained even after denoising. Hence up until this step all (or at least most) of the information in our signal has been preserved.

We define features as partial views of the data. Selecting features according to some criterion amounts projecting it onto a particular view, which usually greatly simplifies the data. A good property of a feature is comparability. That is feature extraction should enable comparison between characters by simple comparisons on the features.

From the curvature representation, we can extract the peaks after the noise-reduction step. For each peak  $i$ , we obtain an  $(index_i, height_i)$  pair. Each  $(index, height)$  pair becomes a feature.

Notice that the number of features is variable according to the number of peaks that the noise-reduction/peak-finding combination will extract. We will construct a model for each character. Then when a new sample is to be identified, we will see how each model explains the set of features in the sample. The model that explains the combination of features best will be chosen as the most likely category that the sample belongs to.

In figure 7 we plot the features extracted for each of the characters.

Notice that the heights of the peaks now go from 0 to  $2\pi$ , since we have applied the transformation of equation 8 to our data, so that high positive peaks and high negative peaks lie close together, while making sure that small negative values and small positive values are separated (see discussion in section 3.5).

Moreover, we have crossed out the peaks occurring within in first 5 and the last 5 points of each sample. This refers again to our discussion in section 3.1. The high derivatives in these points are due to the noise generated when the pen is laid on the table and when the pen is lifted from the table.

Also, we have circled in red the clusters of data that we think a clustering algorithm should pick out. This will help a quicker grasp of the information on this graph. Figure 7a has the regions of interest in an orange square, and figure 7b shows the same figure unedited. We invite the reader to compare the indices and heights of these peaks with those circled in purple in figure 1. For example, we can see all three peaks of character 3, stand out again. The first peak around 10 has a variation of 3 indices in where it occurs, anywhere from 8 to 13. In figure 2, we see this peak occurring at index 11 on the mean character. Hence, this graph provides us not only with the means of the features, but also an estimate of the variation of each feature.

Finally, noteworthy are also the peaks circled in orange. They show that some peaks of 2 and 3 have greater variation in the height of the peaks. This is the artifact of the loop ambiguity that we covered in section 3.4. The height of the peak depends (1) on the presence or absence of a loop, and (2) on the smoothness of this loop. For a 2, the top red circle shows presence of a loop with low curvature,

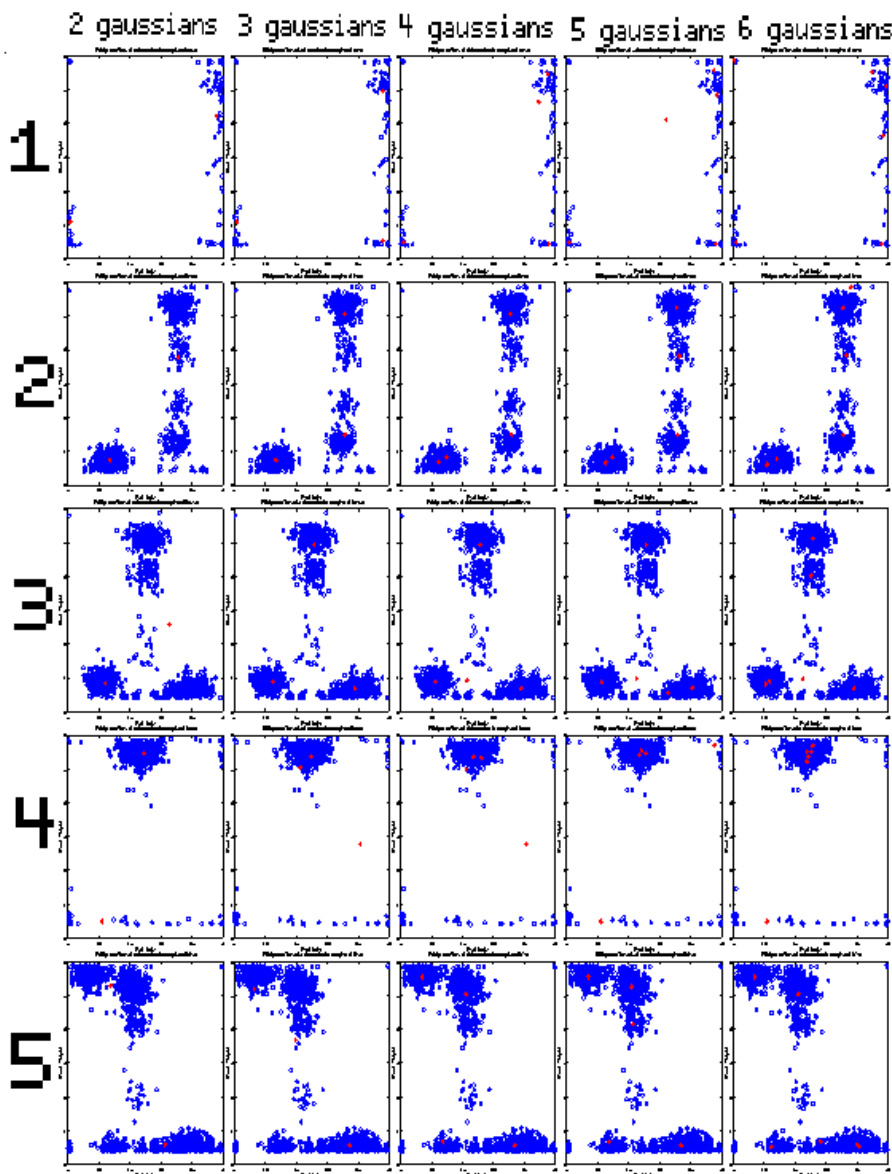
apparently the most likely. The bottom red circle shows no loop and small curvature. Finally the two circles in orange are the high curvature samples, on the top with a loop, and the bottom without. Similar artifacts appear for the crossing of digit 9, and the midpoint of digit 5. These spreads occur for characters that require a sharp angle from the writer, who could stop the pen and hesitate on the curve to follow after that high curvature point. When characters are written rapidly and repetitively, the curve can be smoothed out, include a loop, or a sharp corner. We encourage the reader to analyse similiary the circles for digits 3, 5, and 9.

## Supervised Learning

### Training a mixture for each character

We train a Hidden Markov Model for each of the characters. Each state is made of two components, the index and the height of the peak. Priors for each state are calculated, and with transition probabilities between states. For each character, we train 5 hmm models for different numbers of states (from 2 to 6). Each state can be represented as a gaussian with a height corresponding to the likelihood of the particular center.

We use an EM clustering algorithm to find the centers of the gaussians in order to best explain the data. I didn't use the EM clustering code of problem set 2, but instead I downloaded a matlab implementation of HMM code. (Zoubin Ghahramani <http://www.gatsby.ucl.ac.uk/zoubin/software.html>).



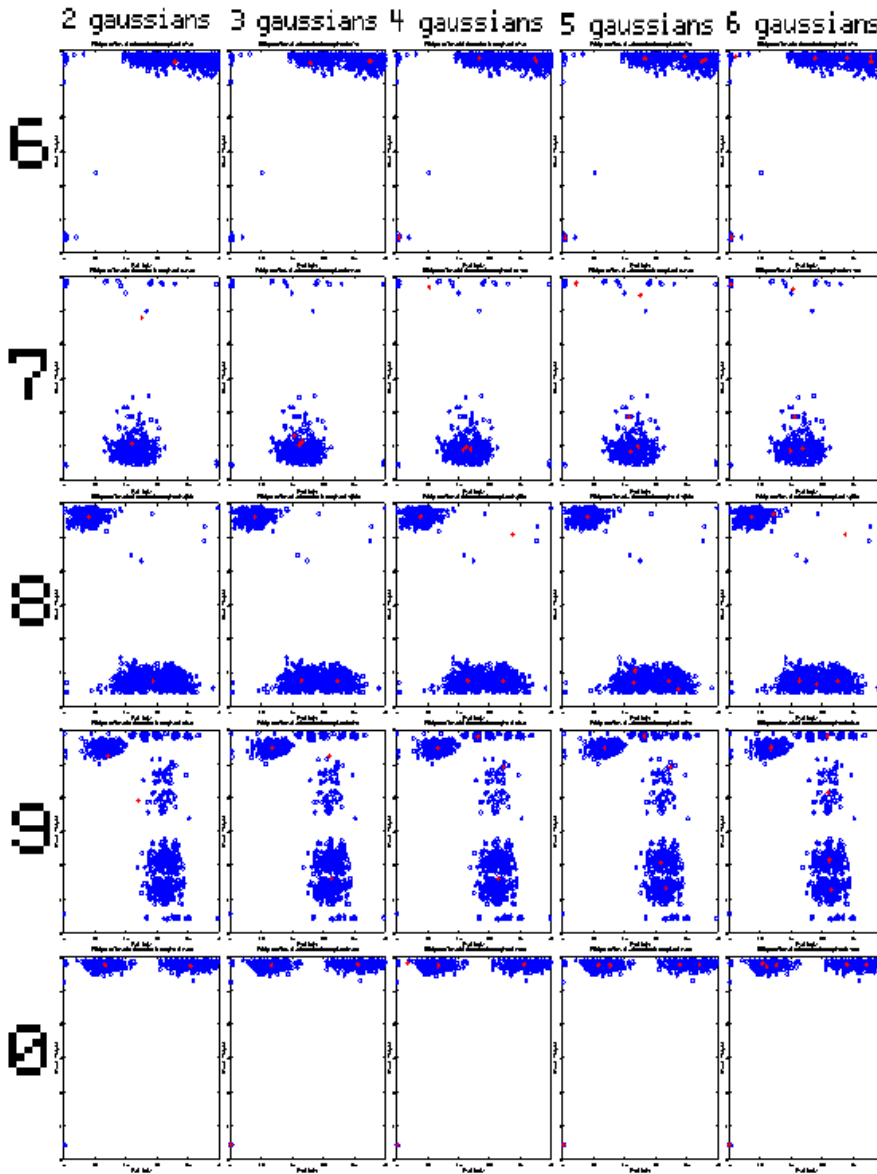


Figure 8 plots the centers obtained after 60 iterations (or after convergence) for each of the characters, and each number of centers. To make the graphs easier to read, again, we have hand drawn the centers in purple, and the countours of the clusters of points around each center are sketched (from figure 7). The original untouched graphs can be found in figure 8b. Sometimes the clouds do not show as well as in Figure 7, since we only drew 60 points, instead of 500, to make the centers visible. In red, we drew for each digit the number of centers that we thought would explain the data best (the one we thought had captured the structure of the data the best). In orange are drawn the ones that actually performed best. We were usually off by one center, but sometimes guessed right.

## Evaluating new data

We first tested for each model  $\theta_j$ , each feature  $f$  of a new sample  $x_i$  independently. The activations from each feature was then combined to form the activation of the character.

$$P(x^i|\theta_j) = \prod_{f=1}^F P(x_f^i|\theta_j) \quad (12)$$

since each model evaluates a mixture of gaussians, this is really

$$(13)$$

$$P(x^i|\theta_j) = \prod_{f=1} \sum_{g=1} P(x_j^i|\theta_{j_g})P(\theta_{j_g})$$

for each gaussian  $g$ , where  $\theta_{j_g}$  are the parameters of the  $g$ th gaussian of the  $j$ th character model.

The independence assumption did a very poor job at explaining the data. The character models that covered most territory were explaining away each peak of each test character, and were attributed most of the points.

This is where the power of the hidden markov model comes in. Transition probabilities are calculated on each of the data points. The longer the sequence, the most likely it is to be correctly identified within the framework of a particular model. The probability now becomes

$$P(x^i|\theta_j) = P(\theta_{j_1})P(x_1^i|\theta_{j_1})P(\theta_{j_2}|\theta_{j_1})P(x_2^i|\theta_{j_2})P(\theta_{j_3}|\theta_{j_2})P(x_3^i|\theta_{j_3})\dots \quad (14)$$

if we know that the first model is  $\theta_{j_1}$ , and that the second is  $\theta_{j_2}$  and so on. If we don't know the order of the models, we have to sum over all possibilities:

$$P(x^i|\theta_j) = \sum_{\theta_{j_1}} P(\theta_{j_1})P(x_1^i|\theta_{j_1}) \sum_{\theta_{j_2}} P(\theta_{j_2}|\theta_{j_1})P(x_2^i|\theta_{j_2}) \sum_{\theta_{j_3}} P(\theta_{j_3}|\theta_{j_2})P(x_3^i|\theta_{j_3})\dots \quad (15)$$

For each character, we evaluate in this fashion every possible configuration for every state within a model. We then choose the maximum activation energy (the highest probability), and we declare the sample as belonging to that character model.

## Results

[Results table](#)

### Good performance for long sequences

After the network was trained, we evaluated the data onto the network. We evaluated the entire data first for each character set, with each model. Using the entire data in batch allowed the HMM model to take advantage of the transition probabilities between states that it was trained to recognize. This yielded a very good performance. The log likelihoods are shown in figure 9.

We expect that for each column the maximum number (the least negative) is on the diagonal. This corresponds to saying that a character is best explained by the model that was built on it. This was in fact almost always the case. Only when we got up to 6 centers the model for 5 was explaining one's data best (because the model for 1 went berserk and couldn't explain anything anymore. Classic case of overfitting. 6 centers are probably too many for a character with no peaks).

We should also expect that the diagonal element is also the smallest element on each row. That is, each model is happiest when it meets its own little fellows. This is not always the case though. The 2-state model for 3 is most happy when it sees a 2 then when it sees itself. We guess that this is because there are not enough centers to discriminate among all the features that make a 2 a 2, and not a 3. As the number of states increases, the model for threes always prefers its buddies than the neighbors.

### Individual character results

Different tests were run for individual characters. The performance was quite good for characters with many peaks, but frankly bad for single-peak or double-peak characters. This goes to show that we should have been more careful in choosing a larger number of features to extract from each character. The peaks are not always there, and the orientation, average length, average peak height, peak length, spaces between peaks could have been good features to augment our dimensionality, in case the number of peaks falls down to 1.

The reason why single-peak characters (such as 7) were often misclassified is that performance for a character test depends how well the different gaussians of a model can explain the sample. In fact, when a simple sample arrives to the system, and a complicated model happens to have a center near the peaks of the sample introduced, then the complex model will be able to explain the data. However, the data will not be able to explain the model. That is, some peaks of the model will not be matching any peaks in the data. Hence our criterion for performance could be changed to include also how fit the model is to the sample, or maybe weight a model by how well it



does on samples overall, and decrease the score of models that explain any sample well.

## Applications and Future work

### Generating characters from a distribution

The HMM models built can be used to constructing new characters. For example, when creating a personal font set, instead of using the same character every time "a" is entered, the system can sample from a distribution trained on the user's own handwriting.

### Handwriting style recognition

Just like different characters can be recognized across users, the distribution of the characters can be used to recognize users. When data is entered in the computer, the writer can be identified as well, among a set of system users.

### Multi-dimensional data

We could include in the model the widths of the peaks, their average values, the spacing between them, etc, instead of working simply with angles. The orientation of the character could also be included as one of the features to train upon (to separate 8's from ∞).

### Wavelet basis tailored to Characters

Instead of using one of the typical wavelet basis (db4), we can envision using a wavelet basis tailored to the character dataset. We can think that this basis will have primitives for a small curve, a large curve, a loop, and that all those will be easily and compactly representable. Finding the right  $h_i$  coefficients for an orthogonal wavelet basis for character recognition could greatly benefit the character recognition community.

## Unsupervised Learning

We postulate that a machine learning algorithm could learn to separate the different characters in features space without any class information. It would be interesting to try and construct such uniform clusters from the data. The algorithm could train upon existing data, and construct meaningful clusters. Trained on existing characters, it could then learn to recognize that a new character it has never seen before is a new symbol, being clustered far from all the other symbols. For data to be as easily separable as possible, we would have increase the dimensionality of our feature space. Right now, this dimensionality varies between 8 (when 4 peaks are found) and 2 (single peak) or even 0 (for some cases of 1. uncorrelated data

### Extension to offline data

To find the (dx,dy) increments from the (x,y) points, we used the fact that the order of the points is meaningful. In fact, instead of (x,y) coordinates, we can think of our data as ( $i$ ,x,y) coordinates where  $i$  is the index of the point (enforcing a logical time in the locations of the pen). We don't use the exact times of each point, but simply the order in which they occur. In fact, since some people write slower or faster, and sometimes hesitate at points in the curve, we postulate that our recognition system should be time invariant. Certainly timing is used in signature recognition and has other useful applications, but in our work, it could be more confusing than useful.

An extension of this work could first translate (x,y) pixels obtained from offline data, and recover the ordering for non-crossing characters, by simply joining nearest neighbors. In digits like 1, 3, 5, 7, 0, the pen never crosses itself, so a unique such ordering can be retrieved (up to a reversing that can occur even in online data, and can be dealt with by establishing a normal order on the strokes, cf. Matsakis).

In self-crossing strokes, such as 4, 6, 8, 9, or a looped 2, recovering the order of points can be tricky. Preserving curvature could be used to disambiguate the order of points in a self-crossing 6 or 8, but this heuristic can sometimes fail, since in handwritten character recognition, the curvature can change drastically from one point to the next. In other applications where physical constraints would not allow such high second derivatives, this approach could be used to recover point order.

## Conclusion

We presented a new compact space for doing character recognition. We showed its application to aggressive compression and recognition. We evaluated a denoising scheme in this space based on recognition performance. We showed how simple features can be extracted from this space and used in recognition and classification. We used an HMM clustering algorithm to learn Gaussian mixture models for each

character, that can classify new characters according to each of the classes. Finally we presented some extensions of the ideas in this work to new applications and new problem domains.

## About this document ...

### Digit Recognition in Curvature Space

This document was generated using the [LaTeX2HTML](#) translator Version 98.1 release (February 19th, 1998)

Copyright © 1993, 1994, 1995, 1996, 1997, [Nikos Drakos](#), Computer Based Learning Unit, University of Leeds.

The command line arguments were:

**latex2html** chars.tex.

The translation was initiated by Manolis Kamvysselis on 2000-06-11

---

*Manolis Kamvysselis*

*2000-06-11*

*Manolis Kamvysselis*

*2000-06-11*