Agency-GP: Agent-Based Genetic Programming for Design

Una-May O'Reilly
Artificial Intelligence Lab.
Massachusetts Inst. of Tech.
Cambridge, MA, 02139
unamay@ai.mit.edu

Peter Testa
School of Arch & Planning
Massachusetts Inst. of Tech.
Cambridge, MA, 02139
ptesta@mit.edu

Simon Greenwold
Massachusetts Inst. of Tech.
Cambridge, MA, 02139
sgreenwold@yahoo.com

Martin Hemberg
Chalmers Univ. of Tech.
412 96 Gothenburg, Sweden
f96mahe@dd.chalmers.se

Abstract

Agency-GP is a prototype for a system using genetic programming and software agents for architectural design exploration. Its software structure is noteworthy for its integration into a high-end three dimensional modeling environment, its allowance for direct user interruption of evolution and reintegration of modified individuals, and its agent-based evaluation of fitness.

1 BACKGROUND AND MOTIVATION

The Emergent Design Group at Massachusetts Institute of Technology (web.mit.edu/arch/edg) conducts research into architectural morphology and the emergent and adaptive properties of architectural form. Our in-house developed software tools exploit models of "natural computation". Natural processes can be viewed both as pragmatic, successful problem solving activities and, as generators of interesting and aesthetic physical or dynamic outcomes. For this reason, they are compelling to abstract and use for inspiration in the implementation of computer software tools for Architecture.

In the context of our work, our natural computation design tools variously employ generative algorithms that respond to environmental properties (Hemberg, 2001, or evolutionary computation. Evolutionary computation makes sense for architectural design both in the role of optimization (not to be discussed herein) and in the role of form design. A key reason is that it can *create* novel, adapted (and often unanticipated) forms without requiring explicit input that directly specifies how to formulate or reformulate a form. In evolutionary computation the architect instead supplies operative criteria (e.g. the state of a form's environment, how a form should fulfill architectural program) or directive information that guides or influences the adaptive process. This provides a means of "saying what is wanted" rather than how it must be designed. It furnishes an architect with a less constrained approach. The inheritance and variation mechanisms of evolutionary computation contribute to apparent form "revisions" that are neither totally random nor totally deterministic. This spontaneity combined with the coherent genealogical process intuitively coordinates with the human design process. Furthermore, evolutionary computation can use specifications of complex and divergent form properties to judge a solution. This implies it has the potential to explore solutions that resolve partially conflicting requirements. In the sense that evolutionary computation is used to pursue a population-based adaptive process rather than optimization, it suits architecture because architects are interested in families of forms rather than finding one absolute best. Also, evolutionary computation allows architects to creatively explore a vastly expanded form space that is too large to investigate by hand.

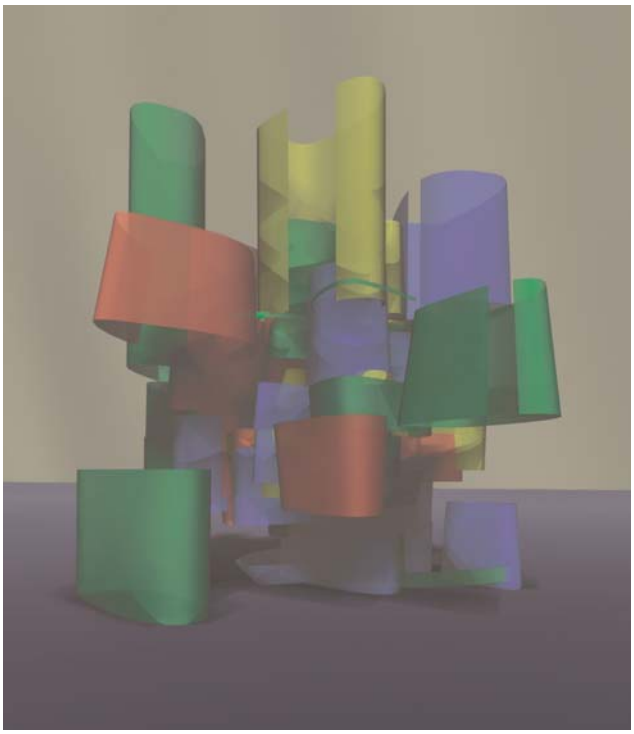

Figure 1. An evolved form from Agency-GP. Shadings denote architectural program. Agents were used to operationalize neighbor constraints, program constraints and height and size constraints.

Agency-GP is the software arm of a larger project called AGENCY (web.mit.edu/arch/edg/agency) currently under way at the Emergent Design group. The project seeks architectural responses to the radical transformation that business organizations are currently undergoing. The pace of organizational change is being driven by the rapid development of commercial technology, global markets and reengineered institutions. This constant need to change gives rise to organizational structures that are no longer stable, but continuously adapting to their shifting environments. Such structures can be said to be "emergent" and describe many of today's commercial and governmental organizations. Vertically structured office buildings no longer provide the model for most businesses. With the advent of widespread use of telecommunications, information technology, and corporate reorganization in the 1990's, new forces are actively reshaping the architecture of office buildings. There is a shift in the United States toward research and development, management and finance, consultancy, and the culture industry, productive activities less prone to standardization and bureaucratization. Driven by the demand to improve office productivity, businesses and organizations have begun to experiment with a variety of alternative officing methods. However, there exist no working models of an intelligent adaptive architecture. The AGENCY project focuses on application-oriented basic research to develop new design software that generatively models the complex interactions of physical space and information technology within emergent organizations.

The Agency-GP tool assists the AGENCY project by providing generative design and test of spatial systems and work environments. Genetic programming has been chosen to address the AGENCY project's challenges because the strengths of the model are well matched to our system's desired characteristics:

First, we are aware of the impossibility of modeling emergent organizations deterministically. To try to design a deterministic algorithm for the creation of workspaces would certainly fall victim to our inability to name every constraint the problem entails. Therefore we look to the stochastic, self-organizing solutions Aritificial Life, and in particular GP affords, to construct solutions that are consistently sensitive to complicated interactions that a user need not explicitly codify.

Second, we are interested in the genetic model's ability to offer a user an entire population of solutions to peruse and potentially evaluate. The process we are involved in is not a simple optimization with a single goal, but has many potential fruitful avenues of exploration. The multi-tracked exploratory process of population evolution provides a designer multiple alternatives with which to interact at any point. Our goal is to include GP as a design partner, offering options that would otherwise not come to light.

The structure of the rest of the paper is: in Section 2 we describe the language (a.k.a. primitive set, set of functions and terminals) central to the GP component. The language defines the universe of executable structures (the genotypes of the system) in Agency-GP and, through the interpretation of a genotype, Agency-GP maps these to surface designs (the phenotypes of the system) which our architects regard as the tool's output. In Section 3 we discuss how we designed an agent-based approach to design evaluation. In Section 4 we present our approach to one central issue arising from using a GP paradigm within a creative design tool such as Agency-GP: how should user interruption, intervention and resumption, (IIR), be handled. We conclude by summarizing the contributions of this tool design effort and suggesting future work.

2 A GP LANGUAGE FOR SURFACES

Much of the success or failure of an application of GP depends on the design of the language that provides the constituitive material for the genotypes or executable structures. There are numerous important considerations to be made. For example, sufficiency, i.e., the language must be sufficiently expressive to yield adequate solution. This is a general consideration for all GP systems. As well, the mapping between genotype and phenotype must be well-matched to the evolutionary operators that will blindly vary the genotype to produce new phenotypes. In the case of Agency-GP the ease with which a genotype could be interpreted into a surface (by the chosen CAD tool Maya) was tantamount. If interpretation took too long, the tool would not meet user expectations. If viewing of any design took too long, the tool again would not fulfill user expectations.

To address interpretive ease, for Agency-GP, our strategy was to use a language that, when used to form an execution sequence, could be directly executed by Maya via its MEL (Maya Embedded scripting Language) facility. MEL is a command line interface to Maya that is more user friendly than its API yet more direct than its GUI. It is intended to provide convenient control of Maya objects to experienced users. The operation sequence portion of the Agency-GP genotype is sent directly to MEL to be interpreted and rendered as a Maya surface.

The individuals in the population are fairly complicated structures, each containing enough information to describe a complete design. We use a combination of Maya and C++ objects for the internal representation. To address language expressivness, Agency-GP uses a language that operates on primitive shapes and combines them with others to build up more complicated designs.

Typically in GP systems, it is difficult or impossible to halt the evolutionary process intermittently to directly modify a phenoype, incorporate that phenotype's new genetic material into the population and then resume. The difficulty arises from languages that contain conditional

structures or use complicated interpretive steps that can not be mapped from phenotype to genotype. The language in Agency-GP was deliberately designed to be non-branching and sufficiently direct under interpretation as to allow for such modification.

A user begins the design process from a Maya scene in which there are one or more closed NURBS (non-uniform rational b-spline) curves selected. These curves should be coplanar but may be of any closed form and may intersect. This initial set of curves acts as the seed for all subsequent evolution. Figure 2 displays an example of an Maya scene with a set of initial NURBS.

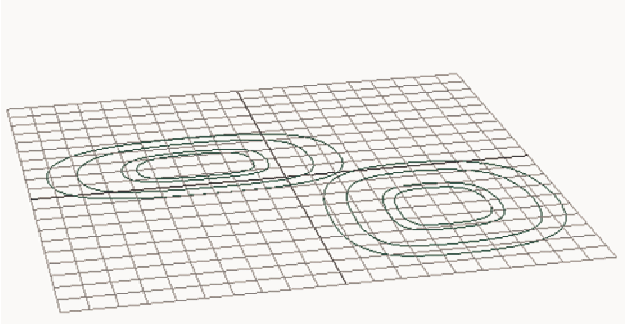

Figure 2. The starting point of Agency-GP is a Maya scene in which any number of NURBS curves are selected.

In the interpreted phenotypic representation, the initial curves from the starting scene will be extruded into space, as shown, for example, in Figure 3.

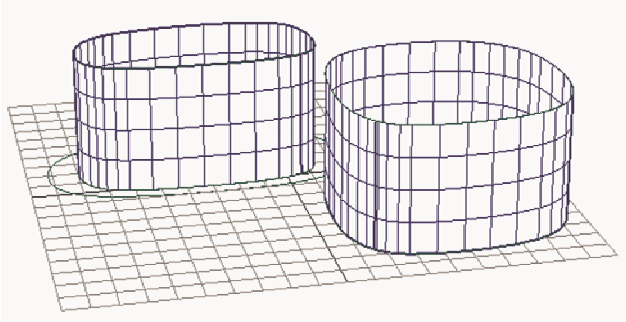

Figure 3. The initial NURBS curves of the Maya scene are extruded into space to create an individual.

When the Agency-GP plug-in is invoked from within Maya, for each member of the population, a C++ object is permanently associated with each NURBS curve in the initial scene. The genotype of the member is a list of these C++ constructs, one per NURBS curve. The construct (or gene) contains evolvable values pertaining to the shape, i.e. each curve is given a height of extrusion and treated as a NURBS surface extending from an evolvable starting height into space, and architectural function of the region the curve encloses The construct also contains an evolvable sequence of operations in our GP language to be applied to this curve. Figure 4 diagrams the internal data structure for an individual genotype in the population.

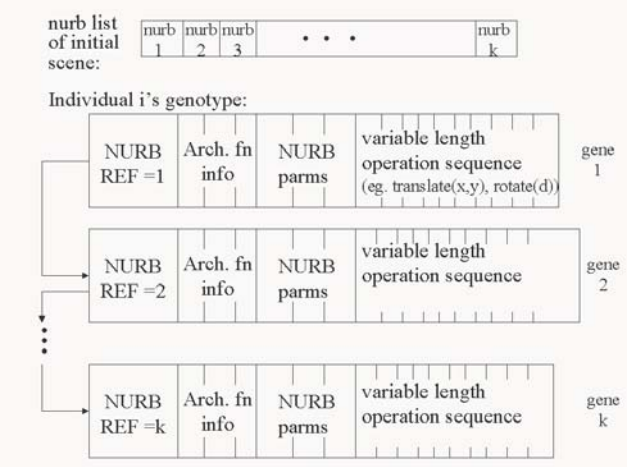

Figure 4. The Agency-GP genotype consists of k genes, each mapping to a different NURBS curve in the initial scene. Each gene encodes parameters to the curve, function information and a varying length sequence of operations that will be executed on the corresponding NURBS curve.

The operations in our language are simple but powerful transformations of these NURBS surfaces: translate, rotate, scale, cut, Boolean intersect and Boolean subtract. The images in Figures 5 through 10 demonstrate these operations of our language when applied to the lefthand NURBS surface from the scene in Figure 3.

These operations form the core commands of our language. Mutation may consist of addition or deletion of an operation, or the change of a parameter. Execution is strictly linear; there is no facility for conditionals or branching. This simple program structure contributes to our ability to implement IIR. We count on these surfaces to intersect with each other in ways we cannot predict. Each Boolean operation we apply-intersection, union, or subtraction-forms a new enclosed surface, which may be assigned its own architectural function.

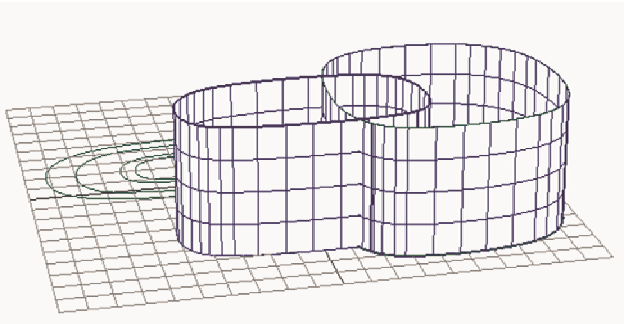

Figure 5. TRANSLATE (X, Y), starting from Figure 3.

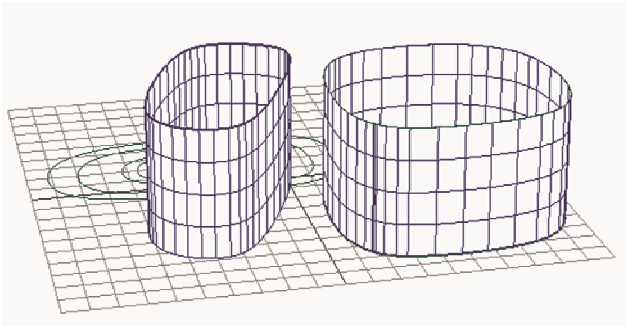

Figure 6. ROTATE (DEGREES)), starting from Figure 3.

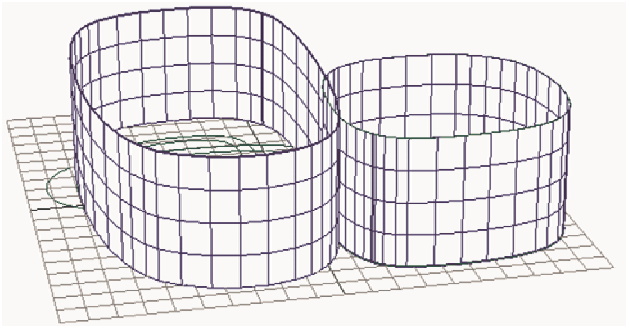

Figure. 7. SCALE (X, Y)), starting from Figure 3.

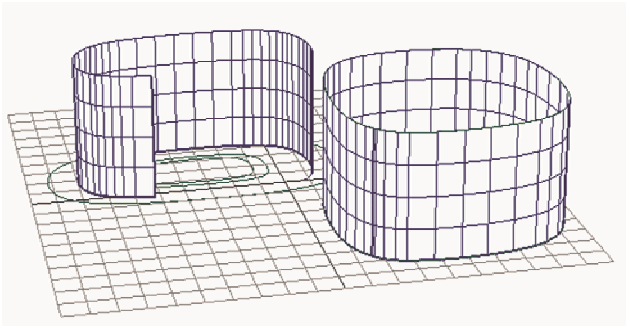

Figure 8. CUT (START, STOP)), starting from Figure 3.

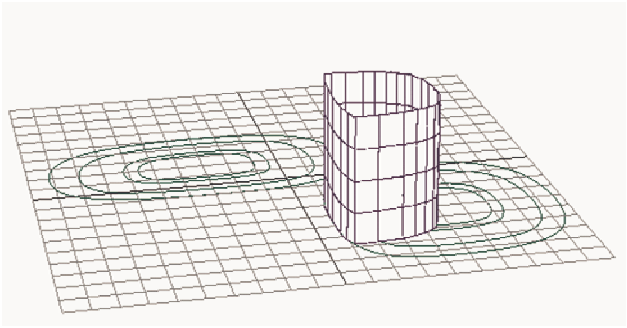

Figure 9. Boolean Intersection), starting from Figure 3.

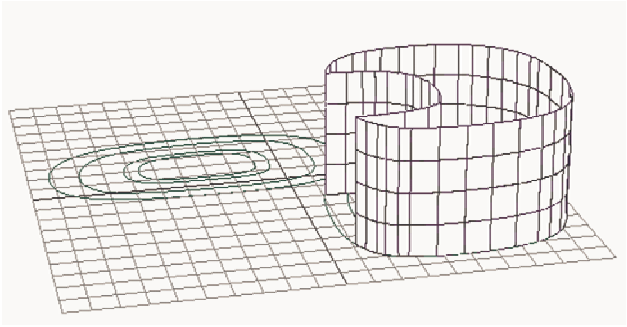

Figure 10. Boolean Subtraction), starting from Figure 3.

That we never directly query or modify the low-level geometry of the Maya objects, but allow Maya to perform all needed transformations and Boolean operations is what makes the language high enough level to be useful for the direct intervention of a designer.

3 DESIGN EVALUATION

We have integrated software agents into Agency-GP to perform design evaluation. Agents are defined differently in different sources, but the general definition we are using is "a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user" (Nwana, 1996). Agents-based models are finding application in fields that require distributed attention and interaction, such as electronic commerce, system administration, network management, and information retrieval (Milojicic, 1999). Agents are autonomous processes that can identify themselves, run concurrently, and interact with each other and their environment. Agents can be supportive of one another, neutral, or antagonistic. There may be multiple instances of the same agent scattered widely or each agent may have an entirely different instruction set. We chose to implement our fitness evaluations by means of distributed software agents inhabiting the candidate designs. Once the designs have been mapped from genotype to phenotype, multiple agents are instantiated to test the designs for various criteria.

Virtually any criterion for evaluation can be coded and function as an agent to our framework. The user can convey general desires, such as "create a high structure over there" or "put a big thing beneath this small one". Or, for a given application, she can specify that workspaces require a certain quotient of natural light or that circulation spaces desire width enough to allow for conversation, etc. . Some agents may represent actual users of the space, while others will be interested in issues such as fire-code compliance, or energy efficiency. Using agent-based evaluation, one is able to model management structures and determine their influence on potential designs. An agent may represent the pattern of a group, its needs for privacy, meeting space and collaborative

surfaces, or it may undertake the concern of management structure or productivity.

Agents individually and collaboratively quantitatively rate the design, and their feedback is incorporated into a measure of overall fitness. The fitness of a design is an amalgamation of several agents' impressions.

The agents give the user high-level control of the fitness function and the evolutionary process. This is a trade-off between the two approaches that are generally found in evolutionary design; a single fitness function expressed in mathematical form (which in fact makes it an ordinary optimization problem) and letting the user act as a fitness function at every step of the evolutionary process (which is very tedious). Each agent is by itself not very complex and could certainly be described as an 'ordinary' fitness function. But the multitude of agents and their interaction gives rise to a complex relationship that can not be captured by a single fitness function. Unless the agent's parameters are chosen in a perverse way, there are innumerable ways to satisfy the agents. This means that the creative and generative abilities that are the strength of the evolutionary design tool are not hampered. The user can adjust the parameters for the agents and it is also easy to change the weight of each agent's impression. In this way it is possible to choose what agents to use or emphasize.

Currently there are five different agents provided by default in Agency-GP. The agents do not experience space in the same way that a designer does. However they each test some concrete (and different) concept concerning space. For example, an agent might consider the concept of absolute position or relative positioning. The combination of different agents leads to a comprehensive concept of space that bears some resemblance to that of a human designer.

Each NURBS curve in Agency-GP has a designated *zonetype* allowing the shapes (i.e. extruded curves) to be divided logically into different groups. Zonetypes are communicated by rendering the surface curves corresponding colors. A group of four agents (one group per zonetype) inspect the shapes of a zonetype and a lone agent operates generally without regard to zonetype. Agency-GP allows different criteria to be desired (for the shapes with that zonetype) by each agent in the group. The weights of each different agent in the group for their linear combination of measures can also be modified. The first zonetype-specific agent rates the sizes of shapes against their desired sizes. The second one considers the quantity of shapes and measures how well they meet a desired proportion. The third agent considers intersections between different shapes which allows the user to stipulate whether two zonetypes should intersect. Finally the fourth agent associates a fitness penalty for any design that violates the constraints of a user-defined bounding box. This provides containment.

The general agent evaluates the height of the entire structure influences the structure to converge towards a desired, user stipulated, height.

3.1 ADVANTAGES OF AGENTS

Agent-based fitness allows for a modular structure for the integration of multiple criteria for fitness. We have abstracted the agent structure so that new agents may be developed and employed for new applications without rewriting the entire Agency-GP system. In fact, one is able to write agents that have non-spatial concerns. For instance, it is possible to implement an agent whose interest is organizing teams of workers. The spatial effects of such an abstracted agent cannot be determined in any way other than to run the system under this agent's selective pressure. Since the descriptive language and representation of the individual designs is spatial, the system effectively translates non-spatial constraints into spatial hypotheses. This agent-based evaluation of fitness is well suited to expressing the conflicting, non-linear, multi-level spatial requirements of emergent organizational structures.

The agent model can be used to layer design intention on top of design intention so that the system is being responsive to more criteria than a human designer could simultaneously manage. Also, with some analysis of agents' satisfactions, it should be possible to discover relationships between one's own design intentions. What are criteria that are hard to satisfy simultaneously? Such information would be useful to both a designer and a tool.

4 TOOL RELATED ISSUES

A GP-based tool provides advantages in the form of generating novel, creative designs. However, GP in its standard form requires tailoring to allow the tool to be optimally responsive and in the control of its user rather than the user being at the mercy of the tool. What in GP has to be changed (or what extensions have to be provided) so that GP is a cooperative interactive assistant rather than a non-interruptible turn-key monolith? We have coined the desired mode of interaction between designer and EC tool as "IIR": Interruption, Intervention and Resumption.

IIR contributes an important element to the relationship between designer and tool. It puts the architect in charge *when she chooses* while ensuring that the overall process (i.e. the process involving designer and tool) is a series of exchanges of control. The architect must be able to influence an ongoing design in order to make it appear the way she wants. She may wish to see what consequences her imposed changes have on the search trajectory through the design space. As the tool develops its outcome, she may wish to change the importance of different desired design properties or explicitly rule out some potential type of design. Ideally, to impose changes, she should be able to use the enclosing CAD/CAM tool within which the tool likely operates. Then, the architect, *without starting over*, would like the tool to proceed again.

IIR as a facility can be broken into its three constituent steps when it comes to implementation. Interruption is a

more or less straight forward mechanism to engineer. It may be controlled via a parameter available at the GUI. For example, in an evolutionary algorithm, the user can direct how many generations run before the search process stops and hands control of the best (or any selected) design over. Alternatively, a user can be given control via a mouse click that controls the tool's steps. [1]

Intervention and resumption are interrelated. There are two means of intervention: indirect and indirect. With direct intervention, the architect actually alters a design and this design is returned to the tool. Resumption involves two tasks if direct intervention has taken place. First, the design changed by the user must be translated into the form of the internal representation and, second, the tool should proceed.

With indirect intervention, the architect does not alter the design (or population of designs). Instead, she influences the computational subprocesses of the tool that, in turn, have an impact on the designs it produces. For example, in an evolutionary algorithm, the fitness function could be altered. If indirect intervention has occurred, resumption involves the tool proceeding with updated parameters.

Interruption privileges are straight forwardly provided via GUI based control in Agency-GP. Resumption is simply enacted with a crude input mechanism such as a key or mouse click which instantiates new values and continues the computation.

In general, handling intervention in a design tool's EA presents more challenges: one first must consider the nature of a design's representations within the evolutionary design tool. Typically a design has both an internal representation (which may be encoded) and an external representation suitable for user presentation and evaluation. The task for direct intervention is to map user enacted changes to the external representation back to the internal representation because it is the internal representation that is used in the evolutionary process.

In some evolutionary algorithms (e.g. simple genetic algorithms, evolutionary strategies) the internal representation describes parameters that are used to elaborate a model (e.g. they are numerical coefficients of a geometric equation). The instantiation of the model with a specific set of parameters is the external representation. This makes mapping a designer-enacted change in the external represention simple if the designer changes something parameterized. Quite the opposite, if she changes any non-parameterized aspect of the design, it is impossible to make the reverse translation.

In other evolutionary algorithms (e.g. genetic programming) the internal representation is actually an executable structure. The executable structure is ``run through'' an interpreter which results in the external design. Executable structures are extremely compelling because they provide more expressive flexibility than model-based parameterization. Direct intervention is hard to engineer in tools that use executable structures due to the interpretive process that the internal representation has passed through in being reformulated as the external design. If the user chooses to stop the tool and intervene with the presented design, there must be a means of backward-translating the updated design into a revised internal representation. Unfortunately the interpretive process is very difficult to reverse. Interpretation is not a one to one mapping so there could be many internal representations that generate an external representation. Which one should be chosen? Alternatively, a change the designer imposes may not even be expressable by the language of the representation and its interpreter.

Once a population has been ranked by fitness, the Agency-GP becomes open to IIR. The entire population of interpreted designs is available for viewing by the user, who has several options. It is possible to indirectly intervene. The user can simply re-rank individuals (i.e. meddle with fitness values relatively) and allow evolution to continue. The user can also control what and how many pre-existing agents will be deployed to evaluate designs, or how heavily to weight each agent's findings. Agents also may have controls of their own which allow a user to direct their activities and thus indirectly readjust the discovery trajectory.

Alternatively, for the first time in our tool suite, direct intervention is available. This is possible in Agency-GP because its language is sequential and non-conditional which facilitates reverse mapping. From the GUI of Agency-GP all operations of the language are permitted to be applied to a displayed and chosen individual. The operations are selected by the user directing changes to the genotype. Then, because the changes to the phenotype are immediately displayed, the user can judge the resulting phenotype and play with what to change very flexibly. Consequently saving the genotype preserves the updated phenotype.

A rhetorical question is whether indirect intervention is better than direct intervention in general. Indirect intervention presents potential for frustration because it forces the tool user to try to influence the process that generates the design rather than allowing direct changes to the design. This 'nudging'' quickly becomes tedious and is often non-intuitive. There are superficial ways to address this phenomenom but it exists precisely because the tools are based on evolutionary or generative algorithm concepts that require (at least) two levels of design representation.

For a tool to facilitate direct IIR, the altered external representation must be sent through the interpretation process backwards to obtain the internal representation that would specify it. However, this backwards process is not simple nor always possible.

For some applications, one may be willing to balance the difficulties of non-intuitivity and awkward usage of indirect IIR with the purpose and benefits of using evolutionary computation. For other applications, it is

[1] Which raises an even thornier question, how do we allow the user to *back up* in a natural computation tool?

simply not acceptable. We think this is an important problem which will determine the ultimate benefit an EA can deliver to an interactive design tool.

5 SUMMARY AND FUTURE WORK

In summary, we have described how an evoluationary algorithm can be incorporated into an interactive design tool. Agency-GP shows that EAs convey distinct advantages to the tool: computationally efficient search, and the generation of novel unanticipated designs. Agency-GP also demonstrates how certain problematic issues of EAs can be overcome in order for the EA not to constrain the power of the tool in other regards. These issues concern automated design evaluation which Agency-GP resolves with an agent-based approach and control mediation.

Agency-GP was developed by a closely interacting team of software designers and architects. At this time, its architecture needs have been anticipated by the architects' specifications. We intend to use Agency-GP in two settings: a student design course and to fulfill some of the goals of the AGENCY project. This will allow us to evaluate its useability and improve it towards wider applicability.

Acknowledgements: We thank the other present and former members of the Emergent Design Group, with particular gratitude to Devyn Weiser. Portions of this paper appear in the Acadia 2000 Proceedings and in a GECCO-2001 Workshop paper entitled "10 Steps to Make a Perfect Evolutionary Design System" co-authored with Peter Bentley.

References

Hemberg, M., U.M. O'Reilly, and P. Nordin, "GENR8: A Design Tool for Surface Generation", in submission to GECCO-2001 Late Breaking Papers track.

Testa, Peter, U.M. O'Reilly, M. Kangas and A. Kilian, " MoSS: Morphogenetic Surface Structure, A Software Tool for Design Exploration", Proceedings of Greenwich 2000 Digital Creativity Symposium, 2000.

Nwana, Hyacinth S., "Software Agents: An Overview," Knowledge Engineering Review, Vol. 11, No 3, (Sept 1996): 1-40.

Milojicic, Dejan, "Mobile Agent Applications," IEEE Concurrency, Vol. 7, No. 3, (July-Sept 1999).