Event Tracker: Question Answering for MIT Events

Ammar Ammar
Erica L. Cooper
May, 2009
6.863 Final Project
1 Introduction

There are many events that happen every day on the MIT campus, from guest lectures and thesis defenses to student group activities, performances, and campus tours. Two sources for information about events at MIT are the MIT Events Calendar (http://events.mit.edu), for general MIT events, and the CSAIL Events Calendar (http://www.csail.mit.edu/events/), for events specific to CSAIL. The MIT Events Calendar lets the user do a basic keyword and date search to find events of interest, but this keyword search only applies to the text of the event description, so if a user wanted to find an event by the name of the speaker or some other metadata besides the title or date, or if they wanted to search for a general topic that may not be explicitly stated in the text of the description, then the simple keyword search would not suffice.

With this in mind, we have designed and implemented a question-answering system for retrieving events of interest from the MIT and CSAIL event calendars. Users can ask questions in natural language about events that are about a certain topic, take place at a certain time, or even retrieve a list of events based on whether they will have food. This system is beneficial to the user because not only does it aggregate events from both calendars, but it also allows the user to formulate his or her queries in natural language, which is more intuitive than keyword search.

2 System Design

The system checks for new events on the CSAIL and MIT event calendars. These new events are then parsed for the relevant fields of data about them, and this information is stored in a database. As user questions come in, they are parsed into queries that can be run on the database.

2.1 Crawling for New Event Notices

Given the large amount of event announcements on campus, and the pace at which these events are added, any information retrieval system has to maintain an up-to-date record of
all these events. To achieve this goal, our system is equipped with a web and email crawler module that maintains these records. All the records are maintained in the database using a Uniform Resource Locator (URL) and a flag to indicate if the event in question has been processed or not. This binary flag is intended as a first level of protection against redundancy in the final semantic database. The crawler can be run as a cron job, to refresh the database regularly.

The events that our system deals with come from the web and a preset collection of email addresses, the information of which are stored in the “email.conf” file. The messages from each email account are fetched and stored in a central location, where they can be accessed later, and their URLs are entered in the database. The crawler maintains the information pertaining to each email address in a configuration file.

The crawler has a nice component for scraping events off event websites using a simple URL-XPath based method. The information about each source of information (events) in RDF like XML files. These files contain the URL of the main page, or the index, the XPath of all the events URLs, and the XPath for any navigation links (e.g. “next” or sibling links).

Extending the crawler is very simple. All we need to do is either add new email addresses to the “email.conf” file, or create new web annotations using a simple FireFox web scraping extension.

2.2 Cleaning Event Pages

Before parsing the events to extract their semantic content, the system needs to make sure that the events are available in a clean and parsable format. This step involves removing HTML tags, substituting some formats for others (e.g. 1:00am for 1:00a), and so on. This step makes things easier for the time expression parser, and speeds up the rest of the process by reducing the amount of text available.

2.3 Parsing Event Notices

Given the page for a particular event, we can extract the following information for that event: the event’s title, the speaker for the event, the speaker’s affiliation, the type of the event, the date and time of the event, the location where it will take place, the event’s host, whether or not food will be served, keywords that are relevant to the event, and the relevant URL for further information about the event.

2.3.1 Regular Expressions

The event notices are semi-structured, so it is easy to extract certain fields of information from event notices using regular expressions. These include the title, information about the speaker, location, host, and relevant URL. The regular expressions are different for the MIT and CSAIL events, so we have separate subclasses with the specific regular expressions for each source. The use of regular expressions is simple but brittle; if we wanted to add a
third source of events, we would have to create a new subclass of event and define regular expressions for it as well.

### 2.3.2 Food

CSAIL event announcements always note the time that refreshments will be served; this makes it simple to determine whether or not a CSAIL event will have food, and so we can use a simple regular expression for this purpose. However, MIT events do not have a structured representation for food, so one must look at and interpret the free-text description of the event in order to determine whether food will be present.

To figure out whether an MIT event would have food or not, we looked at some of the events, decided whether or not they had food, and noted the word that was used to indicate food. We thus compiled a list of food-related words, and we check whether an MIT event announcement contains any of these words to decide whether the event has food.

The list of food words we collected was: “food”, “eat”, “breakfast”, “lunch”, “cookie”, “snack”, “dinner”, and “refreshment”.

### 2.3.3 Event Type

The MIT Event Calendar lists the categories that an event fits into at the bottom of its description; we just use a regular expression to extract those. The CSAIL event calendar does not present the type of the event in such a structured manner. In a similar manner to the food classification described above, we looked many CSAIL event descriptions and determined a few different types of events: these were “lecture,” “seminar,” “thesis defense,” “information session,” and “fundraiser.” Most events were either a lecture, or they had one of these words in its title.

### 2.3.4 Keywords

Not only did we want to extract relevant keywords from the actual text of the announcements, but we also wanted to account for the fact that the user might want to find events about some topic, even if the name of the topic itself is not explicitly mentioned in the text of the announcement. For instance, someone might be giving a talk on semantics, and a user who is interested in language might want to find language-related lectures - such a user would be interested in the semantics talk. However, it is entirely plausible that the event announcement for the semantics talk would not include the word “language” in it, since it would be obvious to a human reader that the talk is within the field of linguistics. If that were the case, then having the system only use words directly from the announcement as keywords would mean that this user wouldn’t find this talk, and they would miss out on it.
As such, we wanted to have a way to bring in additional keywords that were related to keywords extracted from the announcement, even though they didn’t necessarily appear in the announcement itself. Preferably, these words should be more general supercategories of the extracted keywords. To achieve this, we used the Delicious social bookmarking website (http://www.delicious.com/). On this site, users can label their bookmarks with *tags*, which help the user to identify their bookmarks based on topics later on. The idea was that if a user were to tag a webpage with the label of “semantics,” then it would be likely that they would also tag the page with the label of “language,” and that we could use these tags to generate more relevant keywords for events.

To get keywords from the text of the announcement itself, we used a simple TF/IDF metric, which gives a score for a word in a document based on the number of times that word appears in that document, divided by the number of documents in the training corpus that the word appears in. We chose TF/IDF because it is simple and relatively computationally inexpensive as compared to more complicated keyword extraction algorithms, and because it has been demonstrated to be effective, even with as few as around 50 training documents [1]. For CSAIL events, we trained the keyword extractor on the 49 events that occurred in January and February of 2009, and for MIT events, we trained it on the 63 events that took place from January 1st to January 7th of 2009.

After acquiring a list of candidates for keywords for a document ranked by TF/IDF score, we do the following to get at most 20 related keywords from Delicious, along with at most 10 of the highest-scoring TF/IDF keywords:

```r
tf_idfs <- the top three TF/IDF keywords
kw_fqcs <- a map from words to number of times you’ve seen them
kws <- an empty list
while kws.length < 20 and tf_idfs.length <= 10:
   create a list of every (previously unseen) combination of
   three of the TF/IDF keywords in tf_idfs.
   for each of these triples:
      search for them as tags on Delicious.
      get the tags for all of the bookmarks on the first page
      of results.
      for each of these tags:
         update their counts in kw_fqcs.
         add it to kws if it’s not already there.
   append the next best TF/IDF keyword to tf_idfs.
if kws.length > 20: we need to prune it down:
   keep the 20 words with the highest counts in kw_fqcs and
   drop the rest.
   for each element in tf_idfs:
      add it to kws if it is not already there.
```
We use triples of TF/IDF keywords because it was observed that fewer than three gives search results on Delicious that are not constrained enough to give relevant enough keywords, and more than three was observed to be too constrained, very often giving no results at all. We incrementally add each TF/IDF keyword because these keywords become less and less relevant, and if we can get 20 keywords from Delicious by using a small number of only the most relevant TF/IDF keywords, then that is preferable to using more TF/IDF keywords that are less relevant. It was observed that, for many documents, most of the keywords beyond the top 10 were not relevant, thus 10 was chosen as the upper bound. 20 was the number chosen for Delicious keywords for similar reasons. The ordering in which we try the triples favors triples of TF/IDF keywords that are more highly ranked.

It is necessary to take the last step of pruning the keywords list down to 20 because it is possible for a single page of Delicious bookmarks to increase the count of keywords to well over 100. Since our target is 20, we need to prune it down, so we choose the ones that appear with the highest frequencies with the expectation that these will be the ones which represent more general categories rather than tags that are more specific to individual articles.

### 2.3.5 Sepia

Sepia is a framework in which one may express semantic theories, and in which one can compare them, apply machine learning to resolve ambiguity, and even automatically generate new meanings to associate with unseen words. We call it a framework because it combines a novel representation language with accompanying software to aid both manual and automatic semantic lexicon construction. [4]

For the purpose of our project, Sepia can be used to implement a certain representaton for temporal and spatial expressions. It also provides a parser to parse these expressions and produce their equivalent “semantic form”.

Sepia was implemented by Gregory Marton at CSAIL. It is implemented in Guile. A web version of it, with more information, can be found at: http://people.csail.mit.edu/gremio/Sepia/

The lexicons needed to parse the expressions we are interested in have been added to the copy of Sepia that’s used in this project.

### 2.3.6 Date and Time

#### Regular Expression Filtering

For performance purposes, date and time are extracted with regular expressions that are intended to be as generic as possible, given the structure of the current event websites. We are aware that this method is brittle, and is likely to break when presented with different format, but it turns out that processing the whole event text using a specialized semantic engine is a bad idea that most of the available time expression parsers avoid.
That said, once Date/Time strings get extracted from the text, we can parse them using one of the engines mentioned below to obtain absolute Time/Date representations.

Ruby Chronic

Chronic is a natural language (English only) time and date parser written entirely in Ruby. It supports a staggering number of different ways of expressing the date and time. Chronic uses the standard Ruby Time representation, which is an absolute time representation that includes the day of the week, the date, the time, and more details. Chronic can handle expressions like:

```
Today => Wed May 20 22:00:00 -0400 2009
Tomorrow => Thu May 21 12:00:00 -0400 2009
Yesterday => Tue May 19 12:00:00 -0400 2009
Friday => Fri May 22 12:00:00 -0400 2009
This Friday => Fri May 22 12:00:00 -0400 2009
Last Friday => Fri May 15 12:00:00 -0400 2009
Next Week => Wed May 27 12:00:00 -0400 2009
Monday 1:00am => Mon May 25 01:00:00 -0400 2009
1:00am Monday => Mon May 25 01:00:00 -0400 2009
May 15th Midnight => Sat May 16 00:00:00 -0400 2009
```

On the other hand, it cannot handle ranges or expressions like:

```
The Day after tomorrow
The second monday of the month
Christmas
The day before easter
Between 4:00pm and 5:00pm
from 4:00pm to 5:00pm
4:00pm-5:00pm
```

Sepia for Time/Date

As mentioned before, Sepia provides a framework to parse and represent temporal expressions. Unlike Chronic, Sepia can be modified and extended easily by adding more primitive expressions, and rules to that derive the meaning of complex temporal expressions from them. For example, if we have a rule to deal with expressions like “two weeks from this monday”, we can simply add a primitive time expression for “Christmas”, and we will be able to parse expressions like “two weeks from Christmas”. The same logic applies to the other complex expressions that are not parsed by Chronic.

As mentioned before, One nice feature of Sepia is the use of machine learning to get rid of ambiguity, and provide a customized parser. This is important because different people
(i.e. data sources) mean different things by the same time expressions (e.g. next week) For most Time/Date expressions we need for our system, the correct temporal parse has a high probability compared to other non-temporal interpretations of the same string (e.g. the '12' in '12' PM).

After parsing the expressions using Sepia, we extract the Time/Date part of the representation, and convert them MySQL time and date stamps, that can be used for storage and later retrieval.

2.3.7 Location

For the purpose of our system, locations can be represented as simple strings. This is acceptable because the room and building names available on the calendar are usually the most frequently used ones, and there might be no point to use a representation with more details. If event names become insufficient for some reason, a better representation can be added to the system with a little bit of work, thanks to Sepia. The details of the representation are, of course, dictated by the kind of emergent need, but a good representation might want to produce nested location structures that enable the system to recognize that “7-133” is inside building “7”, which is somehow associated with “77 Massachusetts Avenue”, and so on.

2.4 The Database

After all the processing is done, the extracted information is stored in a database called “event_tracker”. The database contains a table called “event_pages”, which contains the index of all the events that has been retrieved by the system, both the processed and the unprocessed ones. The database also contains a table called “events”, which is the main table that stores the aforementioned details of each event. In addition to that, the db contains tables for the keywords and the event types, with links to the main table.

2.5 Question-Answering Interface

In order to answer natural language queries about events, the system needs to translate these queries to the equivalent procedural representation. In our case, this representation can be a SQL query that can be used to obtain the required details.

This parser should be able to recognize valid questions, and identify the relevant parameters provided by the question, in order to use these parameters to construct a specific SQL query. When the domain of the questions is not restricted, this task turns out to be a bit challenging, and more so if we are trying to build or find a parser that fits all kinds of questions.

To solve this problem we can constrain our domain to event announcements and build a specialized question parser that tries to extract Time/Date expressions, locations, and any important parameters, and use them to construct the SQL query. To do this, this parser has to recognize Time/Date/Location expressions, and any modifiers (e.g. after), and use them to create a concise query. For example, a question like:
Q: are there any events tomorrow after 4:00PM?

If we say that tomorrow is (2009-05-10), then the question should translate to something like:

```
SELECT * from events where time > '16:00' and date = '2009-05-10';
```

Note that it is important here to make sure that the conversion from natural language expressions to their equivalent procedural representation (i.e. query parameter) should be consistent with the conversion that was used while processing the events earlier. The failure to meet this requirement means that the answers provided by the system are inaccurate/inconsistent with the actual information.

START

To avoid re-inventing the wheel on the question parsing front, our system uses the natural language capabilities of START, a system developed by Boris Katz of CSAIL. The START system analyzes English text and produces a knowledge base which incorporates, in the form of nested ternary expressions [2], the information found in the text. One can think of the resulting entry in the knowledge base as a digested summary of the syntactic structure of an English sentence. To create this digested summary for time expressions, START uses Sepia, which is one of the available semantic parsers used to parse the original events. This helps us achieve consistent semantics, without having to spend any extra effort.

Because matching occurs at the level of syntactic structures, linguistically sophisticated machinery such as synonymy, hyponymy, ontologies, and structural transformation rules can all be brought to bear on the matching process. Linguistic techniques allow the system to achieve capabilities beyond simple keyword matching, for example, handling complex syntactic alternations involving verb arguments. [3]

In addition to that, START allows our questions to be answered through the same user interface as questions from other topics. This provides us with an opportunity to try our system in the real world, and receive feedback from existing START users. It also allows us to add some manual annotations to improve the performance of the system if errors occur.

3 Code

Our code is written in the Ruby programming language, and we use MySQL for the database. Here is a list of the code files, what they do, and how to use them.

To check out the repository on any CSAIL machine:

```
svn co file:///infolab/svnroot/event-tracker
```
Alternately, you can find all the code mentioned in this document at:
http://csail.mit.edu/ ammar/share/event_tracker

After creating a database called “event_tracker”, you can create the tables of the database using the schemata in “create-db.sql”. After that, you can proceed to retrieve the events, process them, and store them in the database using the following sequence of commands:

   ./crawler/crawler.rb ./templates/calendars.xml
   ./extract_info.rb

To try the code from any of the individual files: type irb in the terminal, and then load ‘filename’.

- **crawler/crawler.rb annotation.xml** — given an annotation for a webpage, you can crawl all the events on that page, and related pages. The crawled events are stored in the database. This is a full fledged web crawler. For more info, look at the documentation in the directory.

- **extract_info.rb** — given URLs for event announcements, makes Event objects out of them with attributes that are the pieces of information that we extract. Example:

   ```ruby
   event_page = EventIndex.new.unprocessed_events[0]
   my_event = EventInfo.make_event(event_page)
   my_event.speaker
   my_event.location
   my_event.food
   ```

   Alternately, calling ./extract_info.rb from the command line extracts the information from all of the newly-posted events and adds them to the database.

- **extract_keywords.rb** — given a URL for an event description, you can get the TF/IDF keywords (get_tf_idf_keywords) and the related Delicious keywords (get_20_related_keywords). Example:

   ```ruby
   url = 'http://www.csail.mit.edu/events/eventcalendar/calendar.php?
   show=event&id=2265'
   get_tf_idf_keywords(url)
   get_20_related_keywords(url)
   ```

- **delicious.rb** — helper for extract_keywords.rb. Gets tags from search results on the Delicious website.

- **date_time_parsing.rb** — helper for extract_keywords.rb. Finds the date on the page and then parses it into the correct format for MySQL.
4 Results and Analysis

4.1 Information Extraction from Event Notices

4.1.1 Food and Event Type Classification

There are some cases where our simple metric of looking for food-related words will give a false positive or a false negative. One example of an event that will be falsely classified as having food is the event “WMBR Presents: My Random Shuffle Is Better Than Your Radio Show,” because it contains the phrase “Breakfast of Champions,” which does not refer to the presence of food at the event, but to the name of a radio show on WMBR, MIT’s radio station. This event would nevertheless be classified as having food because of the presence of the word “breakfast,” which is in our list of food-related words. Events may be classified as not having food if they mention that they will have a particular kind of food that is not on our list, and they do not also mention any of the other words on our list. These cases are rare, however, so this list of food-related words suffices.

In terms of event type for CSAIL events, most events fit into our categories and had the words in the title that would enable us to classify them, but there were a few events that were hard to classify, such as “Short Hands-on Introduction to Fortress,” which was a workshop, but gets classified as a lecture because there is not enough information in the title to decide otherwise.

4.1.2 Keyword Extraction and Generation

We were pleased with the results of keyword extraction and generation using TF/IDF and Delicious tags, especially with CSAIL events, and especially as compared to plain TF/IDF extraction. One example of a particularly good example of keyword extraction is for the CSAIL event “Inference and Learning in Large-scale Relational Conditional Random Fields,” which extracts the appropriate keywords of “machine” and “learning” and “statistical,” but it also comes up with the abbreviations of “ml” and “nlp” as possible keywords, which don’t show up in the original text of the event announcement at all. One example for an event where Delicious tags do not help at all are for the CSAIL event “Accessible Entropy” — the function only returns the top TF/IDF keywords and doesn’t get any additional information from tags. An example of an event where keyword extraction using Delicious tags does worse than just the top TF/IDF keywords is for the event “Non-malleable Extractors & Symmetric Key Cryptography from Weak Secrets” - because the well-known Cryptography characters of Alice and Bob appear in the text, many of the related Delicious tags are also names of people, and these push out more relevant words from TF/IDF like “protocol,” “adversary,” and “key.” Surprisingly, neither use of Delicious nor plain TF/IDF keywords bring up “cryptography” or any of its variants as a keyword, and this indicates that there is room for improvement.
4.1.3 Date and Time Parsing

Our methods for date and time parsing have shown to be reliable - the combined use of regular expressions and semantic parsing to find and create a representation for the time information in these event descriptions works well, determining the correct date and time for all of the events we put into our knowledge base. Choosing the right representation was key—all events take place on one day, and have a starting time, and most events have an ending time as well, and our choice to use a semantic representation for date and time, in the form of MySQL date and time types, was useful in that it enabled us to make appropriate time comparisons.

4.2 Question Answering

START was a good choice for the front-end of our question-answering system because it is easy to use and because it already has a large user base. It does well at determining that a certain question is being asked, even when the user uses different wording to ask the question, for example, active versus passive voice. It is easy to incorporate a new knowledge base into START; to do so, one simply has to write a schema, which is a symbolic representation that describes both the type of information that is available, as well as the generic types of questions that users can ask about it. From these generic question types, START can determine and handle the many different language variations that might occur when a user actually asks the question, and figure out the fundamental structure of the question that the user is asking.

5 Future Work

5.1 Information Extraction

Right now we use very simple methods for determining whether or not there will be food at an event; it would be an interesting project to make a better classifier for this, especially since the cue for whether or not there will be food may be as small as a single word out of the entire description. Similarly, our project would benefit from more general methods for classifying events by type.

Our keyword extraction methods have room for improvement as well. While very often the use of Delicious tags will bring up useful keywords that didn’t appear in the original text, it still fails at finding words for the topic of a talk, as in the example of the talk on Cryptography where “cryptography” was not selected as a keyword. Perhaps the use of morphological stemming would help, since this would ensure that a word and all of its variants get counted together, instead of being considered separate words.
5.2 Question Answering

Due to time constraints, we were unable to integrate our schema for events around the MIT campus with START in time for this project—doing so will be necessary to enable the system to completely work.

6 Conclusion

The methods we used for extracting information from event notices proved to be effective, and we maintained a good balance between regular expressions, which are brittle but fast and simple, and semantic parsing, which gives us useful representations. START is a useful tool for natural-language question answering and, while time constraints prevented us from fully integrating our extracted information about events with START, we believe that natural-language access to information about MIT events would be very helpful for members of the MIT community, and we look forward to getting feedback from users of the system.

References


