Using Explicit Semantic Models to Track Situations across News Articles

Earl Wagner, Jiahui Liu, Larry Birnbaum, Kenneth D. Forbus, and James Baker

Northwestern University
Intelligent Information Laboratory & Qualitative Reasoning Group
2133 Sheridan Road
Evanston, Illinois 60208 USA
{ewagner, j-liu, birnbaum, forbus, baker}@cs.northwestern.edu

Abstract

Online news is a rich information resource for learning about new, ongoing, and past events. Intelligence analysts, news junkies, and ordinary people all track developments in ongoing situations as they unfold over time and initiate queries to learn more about the past context of the events of interest to them. Brussell/STT (Situation Tracking Testbed) is an intelligent information system aimed at supporting this activity. Brussell employs a combination of explicit semantic models, information retrieval (IR), and information extraction (IE) in order to track a situation. It finds relevant news stories, organizes those stories around the aspects of the situation to which they pertain, and extracts certain basic facts about the situation for explicit representation. Brussell uses *scripts* as situation models for the episodes it tracks. Script instances are represented in CycL and stored in the Cyc knowledge-base. Models of ongoing situations can be reloaded and updated with new information as desired.

Introduction

News analysts must sift through massive amounts of data, using perspective gained from history and experience to pull together from disparate sources the best coherent picture of what is happening [Patterson, 1999]. Beyond the levels of simply recognizing entities and the relations holding among them, analysts must make also predictions about situations in the news [Endsley 2001]. Current technology provides some support for these tasks, but in a limited and piecemeal manner.

Our project is aimed at integrating semantic models, information retrieval (IR), and information extraction (IE) technologies to create systems capable of tracking temporally extended situations, aggregating information covering different periods of time and from multiple sources. Our goal is to discover interesting and powerful functional integrations that permit these technologies to exploit each others strengths in order to mitigate their

weaknesses. From the perspective of knowledge-based AI technology, the goal of the project is to extend the reach of such systems into the world of unstructured data and text. Of particular interest is the ability of a knowledge-based system to scale as the size of its input grows. From the perspective of IR and IE, the goal is to leverage the application of inferential techniques in order to achieve significant new functionality. Our goal is *not* to advance the state of the art in information extraction for individual stories *per se*, but rather to use these techniques in a novel context.

This paper begins by describing the functionality and architecture of Brussell/STT. Brussell uses semantic models both to drive retrieval and extraction, and also to organize information about the situation for the analyst. Brussell's performance on a test set of kidnappings as the number of input articles grows is discussed and a more sophisticated syntactic analysis to improve its precision is presented and evaluated. Finally, we describe briefly our proposed design for the next version of the system.

Brussell Functionality: An Example

An example will help illustrate Brussell's functionality. Consider the case of an analyst tasked with identifying factors that predict whether a kidnap victim will escape from his kidnappers. Suppose he recalls reading about the kidnapping of Thomas Hamill. We will see how Brussell simplifies the task of retrieving and organizing the information about this situation.

Entering a Query

The analyst begins in the top-left frame of the Brussell display window by selecting a type of situation to track, and entering identifying information, the *script indicator*, in this case the name of the kidnap victim.

Brussell retrieves a pool of articles from its database, analyzes each to fill out a new script instance and finally updates the display with the instantiated script [Figure 1].

Summary of the Situation

The first important result for the analyst is the high-level summary of the situation. Did he remember the person's name correctly? Did the person in fact escape? How did it happen and what other events preceded it?

In the top-middle frame Brussell shows that it found a kidnapping of the person who did escape. Here, it displays the role names and, more importantly, the specific scenes it found. Each role and scene appears as a button that the analyst can click to inspect. The label of scene buttons displays the scene name, followed by an "x" (times sign) and the number of article references to this scene.

Inspecting Scenes

At this point the analyst wants to go to the beginning of the situation and read about it as it transpired. He begins with the initial abduction event by clicking on first scene button. The date and location of the abduction appear in the top-right frame. In the middle frame are the articles that reference the scene, sorted with the most-recent first. Clicking on one of these articles loads the article with the line or lines that reference the selected scene.

Inspecting Attributes of Scenes and Roles

To be sure he has the date and location the analyst refers to the top-right frame. Here Brussell shows the scene's slot values ranked by frequency. The date "04/09/2004" is the most frequent with 6 votes (indicated by the date followed by "x" then the number of votes). The most frequently mentioned city location is "Baghdad".

If multiple values, such as those for the date of the scene, receive many votes, the analyst may inspect the analyzed sentences manually. Clicking a value loads the middle frame with articles that mention that value, highlighting relevant.

The analyst can also learn more about the kidnap victim himself. Clicking on the "kidnap-victim" role button in the summary loads the top-right frame with ranked values for biographical information about the victim.

In sum, articles are categorized by the scenes they refer to, and by the scene and role data they provide, thus presenting the analyst with an organized means to learn about the situation.

Tracking Situations over Time

Suppose the analyst has read articles and gathered information about one kidnapping and is ready to move on to the next. He can save the current situation to the knowledge base, allowing him to refer back to it later. Saving also enables reasoning systems outside of Brussell to make use of the situation model it has constructed. Clicking a button in the top-left frame saves the script to the Cyc knowledge base.

Brussell supports tracking ongoing situations in addition to situations that have concluded, as we saw here. Instead of creating a new script instance, the analyst can load an existing situation and update it with the stories that have been published since it was last investigated.

Background and Related Work

Recognizing Scripts

Scripts formalize common sense knowledge of ongoing sequences of causally related events, such as going to a restaurant. The events are called *scenes*, and the participants in the events are represented by *actors* that fill *role* slots in the script [Schank 1977]. Scripts were among the earliest high-level representational structures devised for natural language processing. The systems SAM and FRUMP were developed in the 1970s to utilize scripts for understanding. SAM parsed news stories using the conceptual dependency formalism, using scripts to connect the events explicitly described in the story and infer implicit content [Cullingford, 1981]. FRUMP analyzed sentences to fill in shallow "sketchy scripts", situations consisting of causally related events [DeJong, 1982].

Brussell utilizes scripts as situation models to recognize coherence, connect situation descriptions, and perform
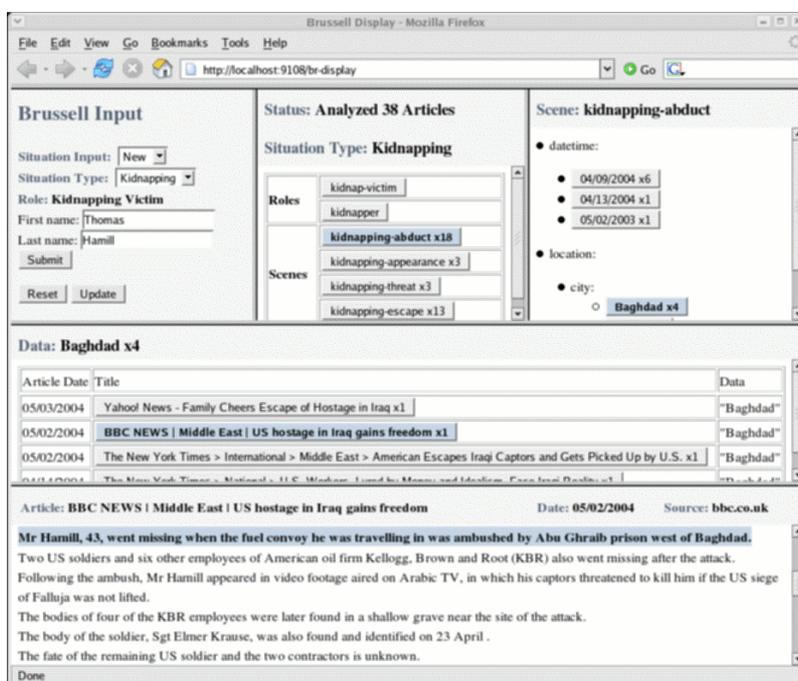

Figure 1. Brussell Display

inference in the spirit of these systems, but is implemented on an entirely different technical substrate utilizing modern IR and IE techniques. Brussell is thus significantly more robust than these early systems, and is able to aggregate information about a script instance from multiple articles.

Recognizing Entities and Events in Text

Later work in NLP continued to focus on selecting and filling in event and entity frames. The Message-Understanding Conferences (MUC) from 1987 to 1998 compared a variety of systems aimed at named-entity recognition and event and entity template filling, among other tasks. Like Fastus, SRI's, entrant in MUC, Brussell uses a finite-state-based approach for matching patterns [Hobbs, 1997].

The Automatic Content Extraction (ACE) competitions continued after MUC in testing the detection of references to entities with texts, and extracting events involving entities and the relations holding among them. However, neither MUC nor ACE tested the recognition of situations spanning multiple events, and the extraction of information from such situations.

More recently, the Proteus system recognizes hierarchically structured event descriptions such as an overall death toll at the beginning of an article followed by descriptions of separate incidents [Huttunen, 2002].

Topic-Detection and Tracking (TDT)

Beginning in 1996, the TDT competitions tested systems in several areas related to organizing news stories about *topics* triggered by seminal *events* [Allan, 2002]. They posed problems including *first-story detection*, recognizing a new topic in a stream of stories, *cluster detection*, sorting incoming news stories by topic and *topic tracking*, in which a system is given some stories about a topic and must determine whether incoming stories are also about the topic.

Systems in the TDT competitions used machine learning techniques to characterize topics and assign stories based on keywords and named entities. This content-independent approach to organizing news articles enabled the creation of robust systems, but suffers from an important limitation. Without an explicit model of the situation, they do not represent the causal relationships holding among individual events and are thus unable to project forward from past events. The closest work to approach this is that of *event-threading* in which sub-topics and their temporal ordering are identified [Nallapati 2004].

Interfaces for Tracking Events

KEDS, and its successor TABARI, code international relations events by recognizing known actors and events using simple syntactic patterns [Gerner, 1996]. KEDS includes a GUI for viewing and editing events extracted from text, but does not organize events by scripts as Brussell does.

Architecture

Scripts for Structure and Semantic Constraints

In addition to organizing information for the user, scripts impose important constraints on whether extracted text is added to the situation representation. First, the identity of the kidnap victim and kidnapper is held to be constant through the course of the script. Thus, biographical information about the victim appearing in any scene that involves him is added to the representation. Second, the script constrains what events the system should attempt to detect, e.g., how the situation ends – with the release, escape or death of the kidnap victim. Voting at the scene level determines the most frequently referenced outcome.

Retrieving and Storing Articles

The system retrieves news articles via RSS feeds from major news sites including BBC News, New York Times, Washington Post, Yahoo! News (which publishes from wire services including Associated Press and Reuters). The content from these articles is extracted and stored in a text indexed MySQL database, providing Brussell with access to new articles daily. Using a database to store articles rather than searching for and retrieving articles with every script query improves speed but also ensures reproducibility of results. Many news sites remove old articles or make them available by subscription only.

Script-Based Query

The user initiates a query by providing a script type and an identifying name, either the kidnap victim name or the city under siege. Upon receiving the script type and name, Brussell retrieves a pool of articles by searching for a set of keywords consisting of the last name and script-specific wildcard terms such as "kidnap* or abduct*".

Analyzing Sentences in Articles

Brussell analyzes articles one sentence at a time. If a sentence has been seen before, as is the case when reading a duplicate article, it is skipped. A sentence is analyzed only if it contains the user supplied script indicator and *scene keywords*, typically verbs indicating the occurrence of a scene. If the system finds both, it runs a simple finite-state pattern recognizer to extract both scene and role information. Patterns are expressed in the form:

    kidnap-abduct -> kidnapper "kidnapped" kidnap-victim
    kidnap-victim -> person
    person -> first-name last-name OR occupation

At each level, the concrete keywords bound the textual region passed to the next level below. The person recognizer looks for slot values within the kidnap-victim range, which is bounded by the "kidnapped" keyword. Some slots including a scene's date and location information may appear anywhere in the sentence. Once a value is found it is added to the appropriate scene or role.

Brussell does not currently employ any methods for resolving anaphoric references. Such methods would clearly improve performance but our focus so far has been aimed more at understanding how to use relatively simple mechanisms within this framework.

Voting

Aggressively merging scene and role references generates spurious data. Brussell was designed to scale well and perform better as it receives more data. To accomplish this, it uses *voting* to resolve ambiguity and reduce comparative noise. As references to scenes, scene slots, and role-filler slots are found, they are added to the script structure with a reference to the article and line in which they appear. These references are also treated as votes. The system ranks values it extracts for slots in the obvious way, i.e., largest number of votes first, then decreasing. The number of votes of the top value relative to the rest is a rough indicator of the system's certainty.

Saving Script Instances to Cyc

Each script instance is stored within a separate Cyc microtheory, the standard technique for representing contexts for reasoning. Slots values are converted to assertions, while votes and sources are saved as comments.

System Evaluation

In order to better understand the strengths and weaknesses of our approach, and particularly the utility of voting, we evaluated the system's performance in analyzing script instances by comparing its scene and role information with hand-coded test cases for a list of kidnappings and in a number of different configurations. Performance was measured using precision and recall for each script according to the following formulae:

$$recall = \frac{N_{\text{correctly filled slots}} + N_{\text{correctly recognized scene}}}{N_{\text{slots of test case}} + N_{\text{scene of test case}}}$$

$$precision = \frac{N_{\text{correctly filled slots}} + N_{\text{correctly recognized scene}}}{N_{\text{filled slots}} + N_{\text{recognized scene}}}$$

Testing Methodology

In October 2004, the Associated Press published the names of 36 foreigners who had been kidnapped in Iraq. With this list, we created test cases for 34 named individuals. Each test case included whether each scene occurred and values for scenes and role slots appearing anywhere in any article about the kidnapping. Testing used approximately 250,000 articles retrieved from April 2004 to February 2006.

The system instantiated kidnapping scripts with roles of the kidnap victim and kidnapper. It found scenes for the initial abduction, threat announcement, (video-tape) appearances, escape, release, kill and/or discovery of the victim's body. The attributes describing the kidnap victim consisted of: first and last name, gender, age, nationality, home town, occupation, and employer. Only the name slot value of the kidnapper organization was extracted.

The system was run with each kidnap victim's name in turn. In comparing the system's output with the test case, we accepted as correct any slot in which it produced the correct result either as its top voted value, or one of its top values if there were multiple equally-highly voted values.

To determine how well the system scales, we also tested its performance on randomly selected subsets of the article sentences in 10% increments.

Evaluation Results

The mean precision for Brussell in these tests was 73% with recall of 59%. The mean F-Measure for Brussell was 66%. For reference, though the tasks are not exactly comparable, the best-performing MUC-7 system achieved 51% F-Measure in event extraction. This comparison is given, and should be taken, with a large grain of salt since the MUC systems were much larger and broader in scope. But they do indicate that Brussell's performance, even with relatively simple mechanisms, was reasonable overall. And they further suggest that the combination of larger models and more semantic constraint, coupled with aggregation of information over multiple articles, e.g., via voting, does provide performance comparable to more sophisticated techniques used without this contextual support. Of all of the slots accepted as correct, Brussell found a single correct value for 83% while the votes for the top value were tied for the remaining 17%.

The following table shows Brussell's performance, broken down into categories for scene selection, scene slots and role slots all of which, for the rest of this discussion we will refer to simply as slots.

Type	Precision	Recall
Scene Recognition	81%	82%
Scene Date	76%	60%
Scene Location	57%	61%
Kidnap Victim Slots	81%	68%
Kidnapper Name	21%	19%

Table 1. Performance on slots values and scene selection

Brussell performed well in recognizing which scenes occurred and extracting scene dates and locations and information about the kidnap-victim. The low recall for kidnapper names may be because the system looks only in sentences that also mention the victim's name, the script indicator. Further, organization names consist of proper nouns, general vocabulary, or a mixture of the two, a traditional difficulty for simple regular expressions. It is expected that even simple anaphora resolution methods will increase recall and more sophisticated named-entity recognition will increase precision.
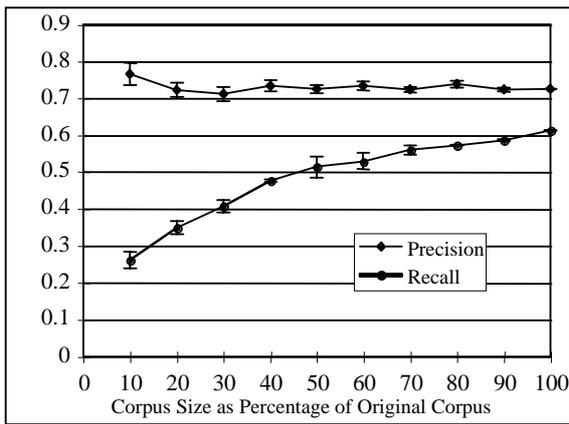
Figure 2. Recall directly related to corpus size

Figure 3. Precision directly related to slot filled time

Figure 2 summarizes the system's performance on increasingly large subsets of the original corpus averaged over four runs (Y-Error bars denote standard error). As expected, recall increases significantly. Surprisingly, since we expected voting to improve results on this measure with more input, there is no significantly increasing trend for precision. We speculated that measuring precision for *all* roles is too coarse-grained. As the system analyzes more articles, it both adds values and votes to "existing," already filled, slots, and finds values for "new," previously unfilled slots – slots which were harder to accurately fill, or for which there simply wasn't enough data yet. In other words, our notion was that the slots that are filled later tended to be harder to fill correctly, or showed up sufficiently infrequently that voting didn't help in a corpus of this size. As a result, "existing" slots may be becoming increasingly correct but be offset by "newly-filled" slots with incorrect values.

To test whether slot correctness did increase with more articles when balanced for this problem, we again divided the pool of articles for each kidnapping into subsets of one sixteenth, one eighth, one quarter, one half and all of the articles, with each larger subset including all of the articles of the smaller subsets. When analyzing a subset, multiple values for precision were calculated: one for all of its slot values, and one for its values for slots that first appeared in each previous subset. Thus after analyzing one quarter of the articles, precision was measured for all slots filled, as well as slots first filled when analyzing one sixteenth of the articles, and those first filled when analyzing one eighth of the articles.

The results for a single run can be seen in Figure 3. Precision for slots under this model trends upward with more articles. The lesson is in retrospect obvious: Voting decreases random error, but not systematic error, and only works if there are enough mentions of a slot value.
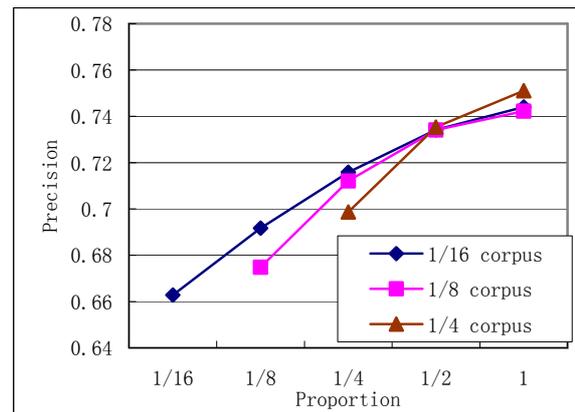
Syntactic Preprocessing to Improve Precision

Many of the false positives in Brusssell's text extraction are due to grammatical structures too complex for the pattern matcher, which assumes simple sentences. Real sentences often contain subordinate clauses, participles, conjunctions, etc. For example, the sentence *"A South Korean hostage threatened with execution in Iraq has been killed, officials in Seoul have confirmed,"* will be matched by the pattern *<kidnapper* "threatened" *kidnap-victim>*, and "South Korean hostage" will be (incorrectly) extracted as the kidnapper.

In order to improve precision, we extended Brussell's simple pattern matcher with a preprocessing module for analyzing and simplifying the sentence syntactically. Prior to applying the pattern matcher, the sentence is parsed using the CMU Link Parser [Sleator, 1991] and simplified according to several heuristics. The preprocessor extracts all syntactic elements relevant to the scene, namely the subject, object, prepositional phrases pertaining to the verb scene keyword. These elements are then restructured to form a simplified version of the original sentence specifically for the scene. For the sentence given above, the link parser generates the following parse tree:

```
(S (S (NP (NP A South Korean hostage)
      (VP threatened
          (PP with (NP execution))
          (PP in (NP Iraq))))
   (VP has VP been (VP killed)))) ,
  (NP (NP officials) (PP in (NP Seoul)))
  (VP have (VP confirmed .)))
```

The scene keywords in this sentence are "threatened" for the threat scene and "killed" for the kill scene. Within the parse tree, "threatened" is a reduced relative clause, so no simplified sentence is generated for this keyword. On the other hand, for the keyword "killed", the simplified sentence is *"A South Korean hostage has been killed"*. It should be noted that the preprocessor also removes "officials in Seoul have confirmed", because the text does

not belong to the minimal S structure of the kill scene. The sentence can now be correctly analyzed and appropriate information extracted, by the pattern matcher.

Through this sort of preprocessing, however, some important information is lost. For example, relative clauses supply information about the referent. To reduce information lost introduced by the preprocessor, we supplement the generic preprocessing mechanism with some heuristics.

One heuristic is to retain relative clauses if they contain certain participles in the simplified sentence. If the participle modifies the subject or the object of the identified scene, and it will not be confused with pattern keywords, the participle will not be eliminated. An example can be found in the sentence *"The kidnappers, calling themselves the Martyrs Brigade, abducted Micah Garen."* The key phrase "calling themselves" precedes an organization's name. Thus the participle will be included in the simplified sentence. In this way the name "Martyrs Brigade" can be discovered and extracted.

Repeating the first test shows syntactic preprocessing increases precision of extraction, while decreasing recall, as we expected [Table 2].

Version	Precision	Recall
Without Preprocessor	72%	59%
With Preprocessor	81%	44%

Table 2. Performance with and without preprocessor

Future Work

Brussell currently searches for script instances only when prompted by the user. The next version will analyze news stories as they are published, adding to existing script instances and creating new scripts.

A weakness of the current version of Brussell is its inability to generate groups of representations in place of individuals. This appears at multiple levels. Kidnappings involving a group of people produced noisier data because biographical information was conflated. Some kidnappings involved repeated scenes, such as multiple video-tape appearances with the result that date information was conflated. Finally, an important new feature of the next version of the system will be to support querying by information other than the kidnap victim's name. For instance, querying by the victim's nationality or the kidnapper's name should be permitted. This is currently unsupported because it requires dynamically generating new script instances in order to handle the fact that these queries will generally match more than one instance.

Other design goals for the next version of Brussell include recognizing patterns in unrecognized text to pass on to the analyst. In some kidnappings, for example, a third party spoke out on behalf of the kidnap victim. These non-scene references to the kidnapping should be provided to the analyst, with the possible result that these events be added as scene types to the script type.

References

Baeza-Yates, R. and Ribeiro-Neto, B. eds. 1999. *Modern Information Retrieval*. Addison Wesley.

Cullingford, R. 1981 "SAM." In *Inside Computer Understanding: Five Programs Plus Miniatures*, Schank, R. and Riesbeck, C. eds. Hillsdale, NJ.: Lawrence Erlbaum.

DeJong, G. 1982. "An Overview of the FRUMP System." In *Strategies for Natural Language Processing*, Lehnert, W. and Ringle, M. eds. Hillsdale, NJ.: Lawrence Erlbaum.

Endsley, M. R. 2001. "Designing for situation awareness in complex systems." In *Proceedings of the Second Intenational Workshop on Symbiosis of Humans, Artifacts and Environment*, Kyoto, Japan.

Gerner, D. and Schrodt, P. 1996. "The Kansas Event Data System: A Beginner's Guide with an Application to the Study of Media Fatigue in the Palestinian Intifada." American Political Science Association, San Francisco.

Grinberg, D., Lafferty, J. and Sleator, D. 1995. "A robust parsing algorithm for link grammars." In *Proceedings of the Fourth International Workshop on Parsing Technologies*, Prague.

Hobbs, J., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M. and Tyson, M., 1997. "FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text'", in E. Roche and Y. Schabes, eds*., Finite State Devices for Natural Language Processing*, MIT Press, Cambridge, MA.

Huttunen, S., Yangarber, R., and Grishman, R., 2002. "Diversity of Scenarios in Information Extraction." In *Proceedings of the Third International Conference On Language Resources And Evaluation*, Las Palmas.

Nallapati, R., Feng, A., Peng, F., and Allan, J. 2004. "Event threading within news topics." In *Proceedings of the Thirteenth ACM international Conference on information and Knowledge Managemen*. Washington, D.C., USA

Patterson, E., Roth, E., Woods, D. 1999 Aiding the Intelligence Analyst in Situations of Data Overload: A Simulation Study of Computer-Supported Inferential Analysis Under Data Overload. Tech Report ERGO-CSEL-99-02, Ohio State University

Schank, R. and Abelson, R., 1977. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Hillsdale, NJ: Lawrence Erlbaum

Sleator, D and Temperley, D. 1991. Parsing English with a Link Grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University.