The Role of Higher-Order Constructs in the Inexact Matching of Semantic Graphs

Michael Wolverton and **Jerome Thomere**
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
{wolverton|thomere}@ai.sri.com

Abstract

Inexact pattern matching using semantic graphs has a wide-ranging use in AI systems, particularly in machine vision, case-based reasoning, and, recently, in intelligence analysis applications. While much previous work in the area has focused on matching simple flat graphs, there is increasing need for and use of complex graphical patterns with higher-order constructs—hierarchical graphs, cardinality constraints, disjunction, and others. This paper reports the results of an experimental analysis of higher-order constructs in graphical patterns and their effect on pattern matching efficiency. The study focuses on two aspects of these constructs—the impact of cardinality constraints on representational power and matching speed, and the benefit of caching hierarchical match results. The analysis shows that both mechanisms provide a significant speedup over conventional flat graph matching.

Motivation

Knowledge representation formalisms based on graphs have a long history in AI (Quillian 1968; Sowa 1984), with applications in many areas, including case-based reasoning (Bunke & Messmer 1994; Cunningham *et al.* 2004), machine vision (Shapiro & Haralick 1981; Sebastian, Klein, & Kimia 2001; Neuhaus & Bunke 2004), bioinformatics (Cook *et al.* 2001), and, recently, intelligence analysis (Wolverton *et al.* 2003; Coffman, Greenblatt, & Marcus 2004; Holder *et al.* 2005). Features of semantic graphical representations that make them appealing, particularly in applications involving the matching or comparison of knowledge structures, include:

- They provide an intuitive visual language for specifying situations of interest.
- Researchers have devised a number of methods for measuring the similarity between two graphs (Foo *et al.* 1989; Poole & Campbell 1995; Bunke 1997; Bunke & Shearer 1998; Zhong *et al.* 2002), supporting the retrieval of structures based on inexact matches.
- Several graphical languages incorporate higher-level constructs on top of "flat" graphs (Sowa 1984; Blau, Immerman, & Jensen 2002; Wolverton *et al.* 2003). These higher-level constructs include hierarchical graphs, disjunction, negation, universal quantification, and cardinality constraints.

One major issue with graphical representations is the efficiency of computing with them, particularly with matching them. Subgraph isomorphism—upon which many semantic graph matching algorithms are based—is a well-known NP-complete problem. In domains such as intelligence analysis, which feature matching over very large relational data sets, efficiency is particularly important.

Higher-order constructs in graphical representations (hierarchy, cardinality, disjunction, etc.) are known to be important for their added representational power. What is missing from the research in this area is an understanding of these constructs' role in the efficiency and scalability of graph-based systems. This paper presents an initial set of experiments designed to measure that role. It focuses on two areas. First, it quantifies the efficiency improvement afforded by precisely representing *cardinality constraints*, compared to approximating them in flat graphs. By "cardinality constraints," we mean restrictions on the number of allowable matches to a subgraph (e.g., "Five or more phone calls"), coupled with the grouping of those matches within a match to a supergraph. Second, it investigates the value of a *caching mechanism* in which subgraph matches are stored and reused throughout the match of a supergraph.

These experiments were conducted using the Link Analysis Workbench (LAW) (Wolverton *et al.* 2003; Thomere *et al.* 2004), which uses an extension of A* to search for the best partial matches to a graphical pattern. It uses a comparison metric based on *graph edit distance* to determine the closeness of the match. The experiments show a significant benefit to both the mechanisms measured.

The remainder of the paper is organized as follows. First we describe the key attributes of graph matching being investigated—inexact matching using graph edit distance, and hierarchy and cardinality in graphs—and their use in LAW. Next we explain the experiments and report the results. Finally, we discuss issues and directions for future work in the area.

Graphical Pattern Matching and LAW

Here we describe the key aspects of graph matching being tested here, and their use in the LAW system. We first give a formal description of graph edit distance as a pattern matching metric, restricting the description to flat (non-hierarchical) graphs. Then we briefly describe higher order constructs in graphical patterns and how the edit distance metric is extended to incorporate them. Last we discuss LAW's method of finding matches to graphical patterns.

Graph Edit Distance

The term "graph edit distance" covers a class of metrics that measure the degree of match between two graphs. Variants have been studied theoretically (Bunke 1997; Bunke & Shearer 1998) as well as applied in a variety of systems. Many of the applications have come in the machine vision community (Shapiro & Haralick 1981; Sebastian, Klein, & Kimia 2001; Neuhaus & Bunke 2004), but the concept has also applied in reasoning by analogy (Wolverton 1994) and other domains.

Let a graph g be a 4-tuple (V, E, μ, ν), where V is a set of finite vertices, $E \subseteq V \times V$ is the set of edges, $\mu : V \rightarrow L_V$ is a function assigning labels to the vertices, and $\nu : E \rightarrow L_E$ is a function assigning labels to the edges. And let a **mapping** between two graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ be a pair $M = (\psi, \varepsilon)$, where $\psi : V_0 \rightarrow V_0'$ is a one-to-one mapping between some of the vertices of the two graphs (where $V_0 \subseteq V$ and $V_0' \subseteq V'$), and $\varepsilon : E_0 \rightarrow E_0'$ is a one-to-one mapping between some of the edges of the two graphs (where $E_0 \subseteq E$ and $E_0' \subseteq E'$).

The metric is based on the action of transforming one graph into another, which in turn is based on edits:

Definition 1 *An* **edit** $g \rightarrow g'$ *is an operation that transforms one graph* $g = (V, E, \mu, \nu)$ *into another* $g' = (V', E', \mu', \nu')$*. The set of all edits is denoted by* Σ*.*

In the LAW system, there are three allowable edit types[1]:

- *Delete Node*: if v is unconnected in g, $g \xrightarrow{-v} g' : V' \cup v = V, E = E'$.
- *Delete Edge*: $g \xrightarrow{-e} g' : V = V', E' \cup e = E$.
- *Replace Node*: $g \xrightarrow{+v'-v} g'$: $V \cup v' = V' \cup v$, E' is E with every appearance of v replaced with v'.

Then a **transformation** T between two graphs g and g' is a sequence of edits $(\rightarrow_0, \ldots, \rightarrow_m)$, where g is the predecessor of $\rightarrow_0$, and g' is the successor of $\rightarrow_m$, and if g_i is the successor of $\rightarrow_i$, then g_i is the predecessor of $\rightarrow_{i+1}$.

There is a one-to-many mapping between transformations and mappings. That is, a transformation T between two graphs g and g' is consistent with a mapping $M = (\psi, \varepsilon)$ iff:

- If nodes v and v' are mapped in M, then there is a Replace Node edit in T replacing v with v'. I.e., $(v, v') \in \psi \Longrightarrow \xrightarrow{+v'-v} \in T$.
- If edges e and e' are mapped in M, then there is a Replace Edge edit in T replacing e with e'. I.e., $(e, e') \in \varepsilon \Longrightarrow \xrightarrow{+e'-e} \in T$.

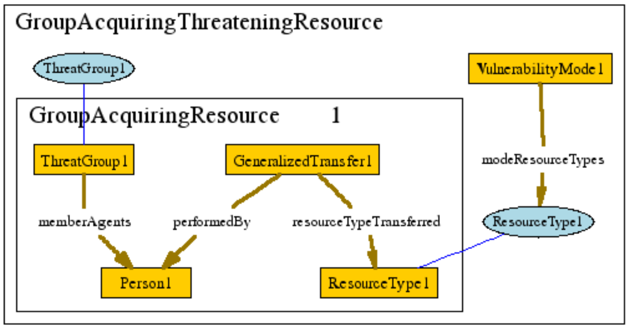


Figure 1: GroupAcquiringThreateningResource—A hierarchical pattern with a cardinality constraint

Definition 2 *A* **cost function** *maps an edit into a cost (a number):* $C : \Sigma \rightarrow \Re$*.*

The cost function value for a particular edit will typically depend on both the edit type, and on the particular nodes and edges being edited. For example, in most LAW patterns, the cost of a Delete Node edit reflects the (user-defined) level of importance of the particular node, and the cost of a Replace Node edit depends on the ontological distance between the two nodes.

It is possible for a cost function to be defined over only a subset of possible edits. For example, it may be desireable to have the cost of a Replace Node edit be undefined if the ontology path distance between the nodes' classes is too great, or if the two classes have no path connecting them.

Definition 3 *The* **cost of a transformation** *is the sum of the cost of the edits in it:*

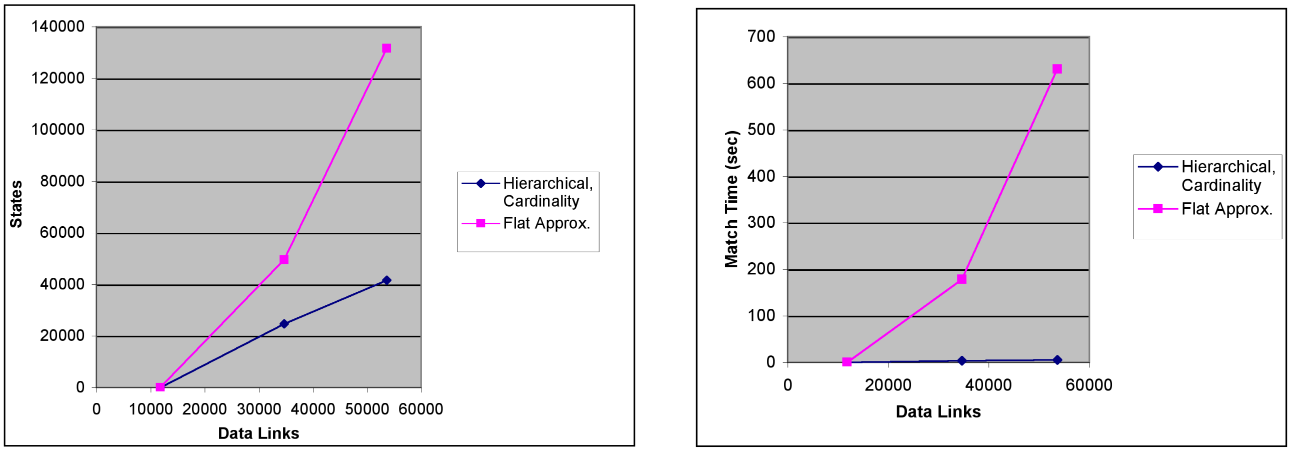
$$C(T) = \sum_{e \in T} C(e)$$

The graph edit distance metric is generally used to find the lowest-cost mapping between two graphs. Here we give a more precise description of what that means:

Definition 4 *The* **least-cost transformation** *of a mapping* M *is the transformation consistent with* M *that has lowest cost. The* **least-cost mapping** *between two graphs* g *and* g' *is the mapping between* g *and* g' *whose least-cost transformation has lowest cost.*

The problem of finding the graph edit distance between two graphs can then be categorized as follows: The graph edit distance between two graphs g and g' is the cost of the least-cost mapping between g and g'.

[1] In the general case, there are three others: node addition, edge addition, and edge replacement. Node addition and edge addition are not relevant in LAW's type of pattern matching (unlike, for example, analogical reasoning), because of the asymmetry between pattern and data: you aren't trying to make the pattern look like the entire data set, only a small portion of it And while edge replacement could be a useful construct in pattern matching, we have not yet found a need for it in practice in our use of the system.



(a)

(b)

Figure 2: Improved efficiency from hierarchy and cardinality on "Group Gets Resources for Mode" pattern, measured by (a) amount of search space explored and (b) match time

Higher-order Constructs

Our design goal for the pattern graph representation is to provide a representational capability that is powerful, but still understandable to a lay-user and reasonably efficient to match. In particular, we want a pattern language that stops well short of the representational power and inferential capabilities of first-order logic or conceptual graphs (Sowa 1984), but still goes beyond the capabilities of simple flat typed graphs. For additional expressive power, we extended the design of the pattern graph representation to include notions of hierarchy, disjunction, and cardinality. By cardinality, we mean specifying information about the number of links, nodes, or subgraphs, e.g. "three or more visits." Figure 1 shows a hierarchical pattern representing a group acquiring a threatening resource. The pattern includes a single subpattern—GroupAcquiringResource—representing a group member carrying out a transfer to obtain a resource. The oval-shaped nodes are *interface nodes*, which are nodes that are shared between the subgraph and its parent graph. The subpattern has a cardinality constraint of "1+", indicating that it will match if one or more transactions involving a given group and a given resource are found.

Pattern Matching Algorithm

LAW's current approach to finding the closest matches to the pattern in the data is based on A* search (Hart, Nilsson, & Raphael 1968). A state in the search is a partial match—a mapping between a subset of the pattern nodes and data nodes, a mapping between a subset of the pattern links and data links, a set of unmapped nodes, a set of unmapped links, and a cost of the mappings so far. The cost is the sum of the delete costs of the unmapped nodes and link, and replacement cost of the node and link mappings, as described above.

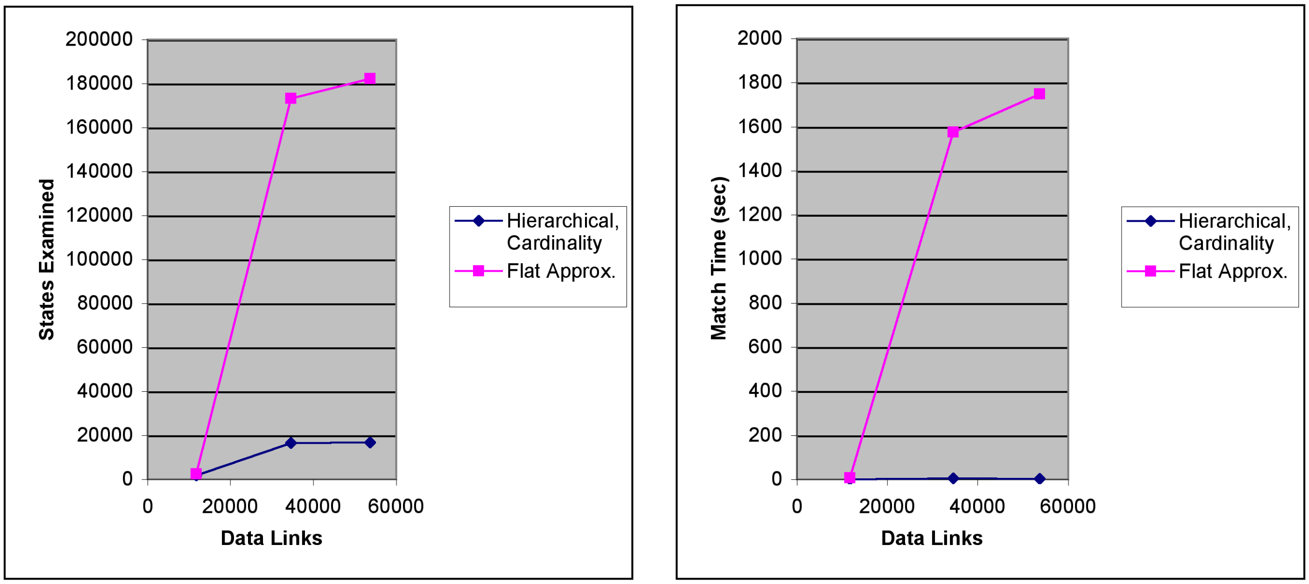
LAW generates start states for the search by selecting the node in the pattern with the fewest legal mappings in the data and creating a partial match for those mappings. It expands a partial match by selecting an unexplored node mapping $(PatternNode, DataNode)$ and generating new mappings for each link adjacent to $PatternNode$ to every mappable link adjacent to $DataNode$. When a pair of links is mapped, the nodes on the other ends of those links are mapped as well. The search selects as the next state to expand the one with the minimum worst-case cost—i.e., the cost of the mappings so far plus the cost of deleting all unexplored nodes and links in the pattern.

The search process is designed to find a good set of pattern matches quickly, and then use those existing matches to prune the remainder of the search. One key asset of the approach is that it is an *anytime* algorithm: at any point during the process the algorithm can return the set of matches it has found already, and that set of matches will monotonically improve as the process continues.

Experimental Results

To evaluate the impact of higher-order constructs on pattern matching scalability, we ran a set of experiments measuring the effect of (1) data set size and (2) various advanced features of LAW—specifically, hierarchy and cardinality—on LAW's match efficiency. The results of these experiments showed encouraging progress toward the goal of solving realistic problems in a reasonable amount of time, and at the same time they indicate areas of needed improvement in future work.

The experiments were designed to measure how LAW's match time changed as various problem characteristics were varied. We used the EAGLE program's data set generator (Schrag 2005) to create data sets of various sizes, where size is measured by number of relations in the data. This data generator creates simulated scenarios involving abstracted



(a)

(b)

Figure 3: Improved efficiency from hierarchy and cardinality on “Hub-and-Spoke” pattern, measured by (a) the amount of search space explored and (b) match time

human activity, including criminal or terrorist activity. It creates simulated people, groups they belong to (“threat” and “non-threat”), resources they acquire, communications between them, etc. Other than data set size, we kept the settings of the generator—observability, noise level, and so on—constant. We ran the set of experiments on a small suite of patterns of various characteristics. Individual experiment descriptions below will describe the specific patterns used. The dependent variables we measured were (1) CPU time elapsed during the matching process[2], and (2) the number of A* search space states examined during the matching process. Our goal was to measure the contribution of higher order constructs in the pattern language toward scalability of the pattern matcher.

Cardinality Results

The first experiment was designed to measure the extra efficiency achieved by adding cardinality to the pattern language and matching capability. The cardinality constuct in the pattern language allows the pattern designer to impose numerical constraints on the number of matches of a subgraph—for example, “three or more meetings.” More important for efficiency, it also provides a way of grouping results of a subgraph match and thereby controlling the combinatorial explosion of possible matches. That is, instead of creating a separate new parent graph match for each subgraph match, the cardinality construct allows LAW to group all matches to a subgraph under a single parent graph match.

In these experiments, we compared the resources used—measured by search space size and time consumed—by the LAW matcher in matching two patterns: (1) a hierarchical graph with cardinality constraints on its subgraphs, and (2) the best “flat” approximation of graph (1). The best flat approximation was determined by inserting N copies of each subgraph into the parent graph, where N is the value of the cardinality constraint on that subgraph.

It is important to note that the flat approximation (2) is truly an approximation to the hierarchical graph (1), and is *not* equivalent, either semantically or computationally. In other words, the fact that we achieved efficiency gains by using hierarchical graphs is not unexpected, and does not represent a breakthrough in complexity theory. Rather, these experiments measure the scalability benefit of giving the user the extra expressive power to represent what he wants. Situation descriptions that include “N or more” occurrences of an event are common, both in the simulated domain used here and in the real world. By giving the user the ability to represent such situations, and by giving the matcher the capability of matching those situations efficiently, we expect to see improved efficiency compared to matching inexact approximations.

Figure 2 shows the efficiency gain from hierarchy and cardinality on one pattern, representing a threat group with two or more members each acquiring a threatening resource. The hierarchical pattern contained a single subgraph with a “2 or more” cardinality constraint. The graphs show the change in search efficiency as the data set size grows, with the smallest data set size containing around 15,000 links and the largest

[2] The runs were with the LAW pattern matcher implemented in Allegro Common Lisp v6.2, running under Solaris on a SunBlade 1500.

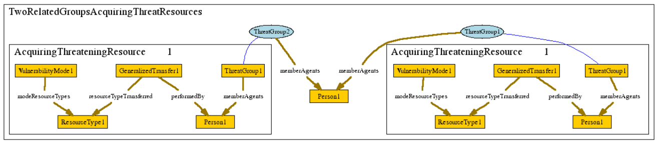


Figure 4: “Two Related Groups Acquiring Threat Resources” pattern

containing around 75,000. Figure 2a) measures the number of search states explored, which was reduced by a factor of more than three using hierarchical graphs on the largest data set. Even more encouraging is the amount of time saved by hierarchical patterns, shown in Figure 2b. Hierarchical patterns with cardinality improved match time by over two orders of magnitude—from 630 seconds to six.

Figure 3 shows the same measurements for another pattern (and its flat approximation). This pattern represents hub-and-spoke communication—two or more phone calls from members of a threat group to another “hub” group member. Like the “Group Gets Resources for Mode” pattern of Figure 2, the hierarchical version of the hub-and-spoke pattern contains a single subgraph with a “2 or more” cardinality constraint. The efficiency gain with this pattern was even more dramatic than that shown in Figure 2. Figure 3(a) shows a search space reduction on this pattern of over an order of magnitude—from 180,000 states to 16,000—and Figure 3(b) shows a match time reduction of almost three orders of magnitude—from over 1,700 seconds to less than three.

The dramatic difference between the search space reduction and the search time reduction—the (a) and (b) graphs in each figure—is an interesting phenomenon, and one for which we do not have a definitive explanation. Our current hypothesis is as follows. State expansion in LAW’s A* search involves copying the parent state. When the patterns are hierarchical, the pattern representation is more compact—since each subgraph is represented in a parent graph as only a pointer, rather than a complete list of all nodes and links in the subgraph—and therefore the copying operations per state expanded are smaller.

Caching Results

The second set of experiments measures the value of caching subgraph matches in hierarchical pattern matching. We designed a mechanism in the LAW matcher that caches subgraph match results during a match of a parent graph. Subgraph matches are cached indexed by

- The subgraph being matched, and
- The mappings of the subgraph’s interface nodes.

Once a subgraph S with a given set of interface node mappings $< m_1, \ldots, m_n >$ is matched, the results for any future attempts to match S with $< m_1, \ldots, m_n >$ will be retrieved and returned from the cache.

Our hypothesis was that caching would be especially useful for patterns that contain multiple copies of the same subpattern. Such a pattern is shown in Figure 4. This pattern represents two different Threat Groups, related through a common member, each acquiring a threatening resource. Matching this graph will require multiple attempts to match the AcquiringThreateningResource subgraph for each Person, and caching will eliminate all but one of those attempts.

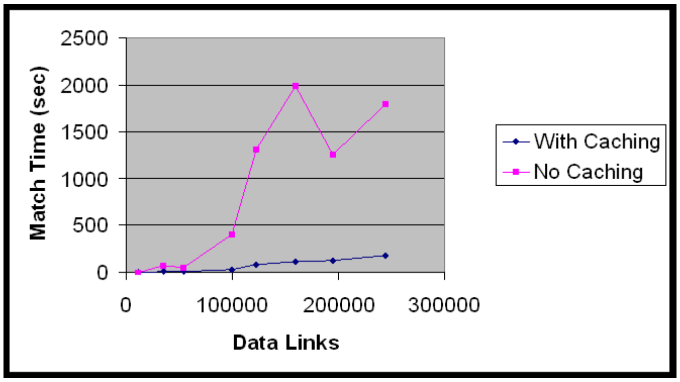


Figure 5: Effect of caching as data set size grows on “Two Related Groups Acquiring Threat Resources” pattern

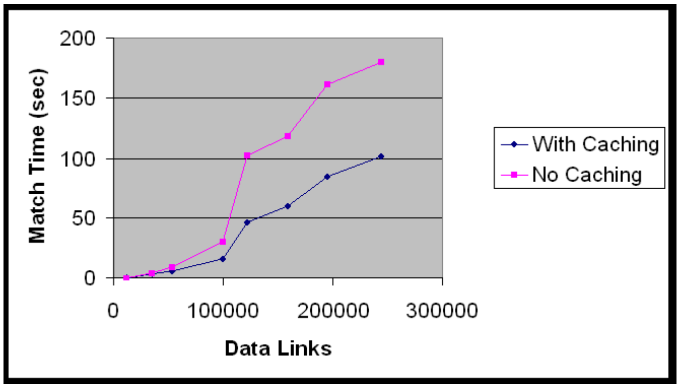
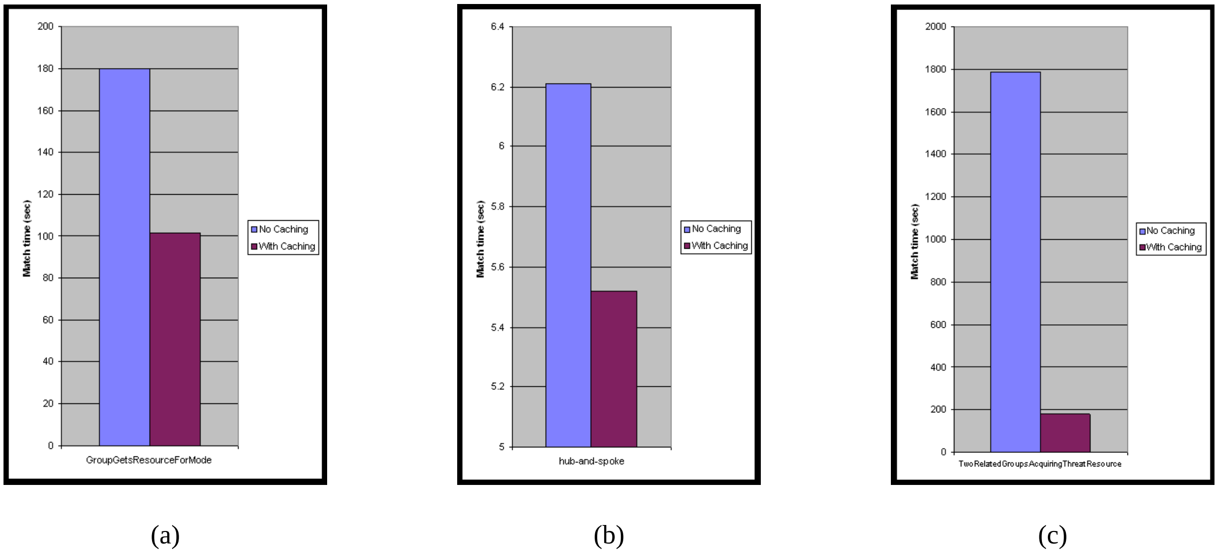


Figure 6: Effect of caching as data set size grows on “Group Gets Resources for Mode” pattern

As we expected, Figure 5 reduces match time dramatically for Figure 4’s pattern. For the largest data set tested, caching reduced match time by an order of magnitude, from 1,800 seconds to 180, and the time savings were even greater for some smaller data sets.

However, we did not expect the results shown in Figure 6. The pattern that generated those results—“Group Gets Resources For Mode”—is a pattern with a single subpattern, and one for which we expected little or no repetition. Our hypothesis was that caching would have little impact; in fact, we thought it would be possible that the overhead associated with caching could cause the runs with caching to be slower than the ones without. Instead, Figure 6 shows that caching did have a significant positive effect even for this pattern, cutting match time roughly in half for all data set sizes tested.

Figure 7 summarizes the caching results, showing the benefit of caching for all three patterns tested on the largest



(a) (b) (c)

Figure 7: Summary of caching results: benefit of caching on largest data set for (a) "Group Gets Resources for Mode", (b) "Hub-and-spoke Communication", and (c) "Two Related Groups Acquiring Threat Resources" patterns

data set tested.

Discussion and Future Work

The experiments reported here represent only a first step in understanding the role of higher-order constructs in graph matching efficiency. There are many open areas for future work, including:

- Further experimentation involving a larger test set of patterns, data sets with different characteristics, and/or different independent variables. One area in which these experiments only scratch the surface is the degree to which inexact matching (vs. exact matching) influences the results.

- Further exploring mechanisms for caching intermediate results. One topic to explore here is optimizing the level of caching that goes on within a pattern match. One can imagine a continuum of degrees of caching, from no caching at all to the caching of a match at every state in the seach (which essentially amounts to a dynamic programming approach to the problem). The experiments of the previous section strongly suggest that the no-caching side of the continuum is suboptimal. But can we identify (characteristics of) an optimal point on the continuum? And if so, can an optimal caching scheme be automated?

- Investigating the degree to which the results here depend on LAW's particular pattern matching algorithm. It seems likely that caching and (especially) precise cardinality representation would have some benefit in alternate approaches to A*—including dynamic programming and genetic algorithms. But how much?

Acknowledgements

This research was supported under Air Force Research Laboratory (AFRL) contract number F30602-01-C-0193. Thanks to Brian Gallagher and Tina Eliassi-Rad, who identified an error in the workshop version of this paper.

References

Blau, H.; Immerman, N.; and Jensen, D. 2002. A visual language for querying and updating graphs. Technical Report 2002-037, University of Massachusetts Amherst Computer Science Department.

Bunke, H., and Messmer, B. 1994. Similarity measures for structured representations. In Wess, S.; Althoff, K.; and Richter, M., eds., *Topics in Case-Based Reasoning*. Springer-Verlag.

Bunke, H., and Shearer, K. 1998. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters* 19:255–259.

Bunke, H. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters* 18:689–694.

Coffman, T.; Greenblatt, S.; and Marcus, S. 2004. Graph-based technologies for intelligence analysis. *Communications of the ACM* 47(3).

Cook, D.; Holder, L.; Su, S.; Maglothin, R.; and Jonyer, I. 2001. Structural mining of molecular biology data. *IEEE Engineering in Medicine and Biology* 20(4):67–74.

Cunningham, C.; Weber, R.; Proctor, J.; Fowler, C.; and Murphy, M. 2004. Investigating graphs in textual case-based reasoning. In Funk, P., and Gonzalez Calero, P., eds., *Advances in Case-Based Reasoning (Lecture Notes in Artificial Intelligence, Vol. 3155)*. Springer-Verlag.

Foo, N.; Garner, B.; Tsui, E.; and Rao, A. 1989. Semantic distance in conceptual graphs. In *Proceedings of the Fourth Annual Workshop on Conceptual Structures*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Holder, L.; Cook, D.; Coble, J.; and Mukherjee, M. 2005. Graph-based relational learning with application to security. *Fundamenta Informaticae* 66(1–2).

Neuhaus, M., and Bunke, H. 2004. A probabilistic approach to learning costs for graph edit distance. In *17th IEEE International Conference on Pattern Recognition*.

Poole, J., and Campbell, J. 1995. A novel algorithm for matching conceptual and related graphs. In *Conceptual Structures: Applications, Implementation and Theory—LNAI 954*. Springer-Verlag.

Quillian, M. 1968. Semantic memory. In Minsky, M., ed., *Semantic Information Processing*. MIT Press. 227–270.

Schrag, R. 2005. A performance evaluation laboratory for threat detection technologies. Submitted for review.

Sebastian, T.; Klein, P.; and Kimia, B. 2001. Recognition of shapes by editing shock graphs. In *IEEE International Conference on Computer Vision*.

Shapiro, L., and Haralick, R. 1981. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3:504–519.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.

Thomere, J.; Harrison, I.; Lowrance, J.; Rodriguez, A.; Ruspini, E.; and Wolverton, M. 2004. Helping intelligence analysts detect threats in overflowing, changing and incomplete information. In *2004 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*.

Wolverton, M.; Berry, P.; Harrison, I.; Lowrance, J.; Morley, D.; Rodriguez, A.; Ruspini, E.; and Thomere, J. 2003. LAW: A workbench for approximate pattern matching in relational data. In *The Fifteenth Innovative Applications of Artificial Intelligence Conference (IAAI-03)*.

Wolverton, M. 1994. *Retrieving Semantically Distant Analogies*. Ph.D. Dissertation, Stanford University.

Zhong, J.; Zhu, H.; Li, J.; and Yu, Y. 2002. Conceptual graph matching for semantic search. In *Proceedings of the Tenth International Conference on Conceptual Structures (ICCS 2002)—LNAI 2393*.