Software Supported Pattern Development in Intelligence Analysis*

Michael Wolverton and **Ian Harrison** and **John Lowrance** and **Andres Rodriguez** and **Jerome Thomere**
Artificial Intelligence Center
SRI International
333 Ravenswood Ave
Menlo Park, California 94025
<*lastname*>@ai.sri.com

Abstract

Intelligence professionals work with incomplete and noisy data. Their information needs are often hard to express, and almost impossible to get right the first time. This paper describes the GEM pattern language for encoding analysts' information needs in graphical patterns, and its use in the Link Analysis Workbench (LAW) system to find inexact matches to those patterns in large relational data sets. The LAW user typically interacts with the system through a cycle in which the user (1) creates an initial GEM pattern corresponding to his information need, (2) uses the LAW matcher to retrieve a collection of matching episodes in the data, (3) revises the pattern based on the shortcomings of the matches, and (4) repeats the process until the revised pattern is returning the right data. The pattern language and the system are designed to facilitate the user in quickly traversing this cycle.

Introduction

Information gathering for intelligence analysis requires flexibility, on the part of both the analyst and the information gathering tools he uses. The data sets from which information is collected are typically noisy and incomplete. Situations of interest to the analyst may differ only subtly from common and irrelevant situations. And in many cases the analyst may only be able to describe the situation of interest at a high level or with a limited amount of precision, because of imperfect knowledge about the terminology, organization, or content of the data he is searching.

For these reasons and others, information gathering is rarely a one-shot operation. Instead, the process is generally an evolutionary *cycle*, where the pattern of interest is constructed and then repeatedly refined based on results returned from the data. The analyst is heavily involved at all stages of the cycle. Supporting this cycle poses technical challenges for the tool developer, both in designing a pattern language flexible enough to describe both very specific and very general match criteria, and in producing a system that allows the analyst to define and refine patterns and interpret results quickly.

This paper describes the pattern development cycle as supported by the LAW system (Wolverton *et al.* 2003). LAW features several characteristics that are important for this cycle:

1. An intuitive pattern language based on *semantic graphs*.
2. A simple *similarity metric* that supports the retrieval and ranking of inexact matches.
3. A pattern editor that supports easy editing of patterns, and a pattern library that allows users to construct complex hierarchical patterns out of simpler, previously-defined ones.
4. A match display interface designed to allow the user to understand at a glance the quality and content of a match to a pattern.

We begin by illustrating LAW and its support for the pattern development cycle through an extended use case of intelligence gathering using the system. We then describe LAW and its pattern language, GEM, in more detail. Finally we discuss some key issues and areas for future work.

The Intelligence Analysis Cycle in LAW

To demonstrate LAW's support for the pattern development cycle, the next section will present an example from the domain of the EAGLE challenge problem simulator (Schrag 2006). Here we give a brief summary of that simulated domain. The challenge problem is to support an analyst in identifying, within a large database, instances of planned attacks by terrorists on infrastructure elements, exploiting known vulnerabilities. The goal is to infer from the data as much detail about the planned attack, before the attack happens, which includes the people involved, the target, the method to be used and the planned steps involved. In addition evidence found in the data to support the hypothesis that an attack is being planned should also be reported, such as available transaction data, like resource acquisition (e.g. buying dual-use materials), as well as any relevant SIGINT data (e.g. telephone intercept records).

The simulated domain is made up of ordinary people and terrorists (people are termed Agents in the domain). Everyone is a member of one or more groups. Terrorists are members of one or more *ThreatGroups* (as well as potentially being members of *NonThreatGroups* too.) All people in the domain have a number of *capabilities* (e.g. Flying a plane, Driving a large truck) throughout the length of the

*This research was supported under Air Force Research Laboratory (AFRL) contract number F30602-01-C-0193.

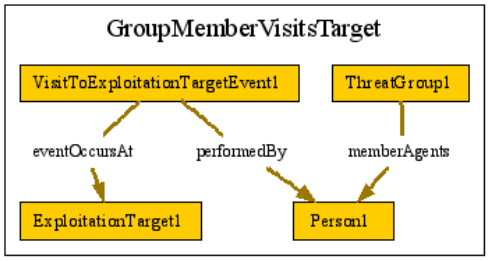

Figure 1: GroupMemberVisitsTarget—a basic graphical pattern

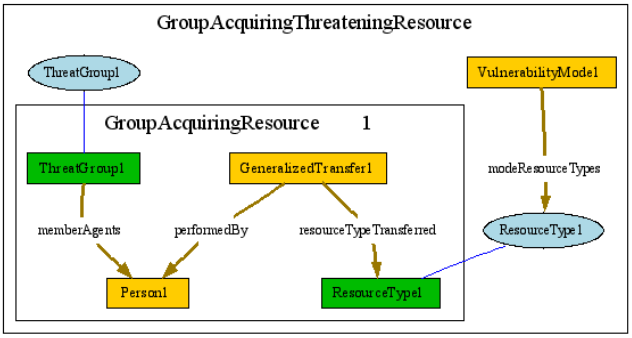

Figure 2: GroupAcquiringThreateningResource—A hierarchical pattern with a cardinality constraint

simulation. Some of the capabilities are reported, some are not. People make purchases (acquire *resources*) throughout the simulation (e.g. ammonium nitrate fertilizer, car rental, hotel reservations), for which the transaction records may or may not be available. Agents communicate with one another and amongst group members, to plan events (both benign and threat), possibly leaving a transaction trail, and make visits to infrastructure items (termed *Targets* in the domain).

Groups (both ThreatGroups and NonThreatGroups), have designated methods (*modes* in the domain ontology) which they apply at a target. Modes are sets of capabilities and resources. Targets have associated modes too, which reference ways in which resources and capabilities can be applied at a target. Some of these modes are legitimate use of the target (e.g. parking a rental car in an underground car park above a large public building, taking a boxknife to a mall). These are termed *ProductivityMode*'s. Other modes however, called *VulnerabilityMode*'s, are only exploited by ThreatGroups, (e.g. taking a bomb to a building).

The task for the analyst is to sift through a large database of records and try to identify early warning signs that indicate that a ThreatGroup is planning an attack on a target through a particular VulnerabilityMode, using some of the capabilities and resources of its members. The task is made more complicated, because the data is not perfect. For instance, not all ThreatGroup's are known and neither is their complete membership nor their VulnerabilityMode's. Some of the data is also corrupted. In addition, some of the resource and capability application data is indicative of both legitimate behavior as well as illegal behavior.

Use Case of the Pattern Development Cycle in LAW

In contrast to systems that are designed to support link analysis in a single pass, LAW is designed to support a pattern development cycle in which the analyst creates, evaluates, and refines the pattern repeatedly. This section describes this cycle through a hypothetical use case for LAW, in which an analyst searches for situations of interest within a dataset created by the simulator described in the previous section. The data set in this example contains approximately 130,000 nodes and 330,000 links, and all the results discussed below represent the results of actual runs of the LAW matcher against this data set.

The analyst in our scenario has been using patterns designed to look for threatening group activity. A group must meet several conditions to carry out an attack on a target. Figure 1 shows one. It represents a group member visiting the target. Figure 2 shows another. It represents a group acquiring a threatening resource—i.e., members of the group make one or more transactions acquiring a resource that can be used to exploit a VulnerabilityMode. The larger pattern the analyst uses to detect threatening activity (not shown) looks for a single group meeting all the conditions. It is constructed using LAW's graphical pattern editor (an example screenshot of which is shown in Figure 3) by combining and connecting these (and other) smaller patterns.

To complement the search for threatening group activity, the analyst gets the idea of looking for a slightly different kind of scenario: one where two threat groups cooperate to carry out different portions of an attack. The idea he has is that the two groups will cooperate through an *intermediary*—a person who is a member of both groups. He uses the pattern editor and the pattern library to construct a new pattern out of the same primitive graphs from the single-group pattern; the only change is to add a Person node for the intermediary and have two separate group nodes instead of one.

The result is shown in Figure 4. The pattern represents two groups potentially cooperating—one group makes one or more visits to a target, and the other acquires two or more resources necessary to attack that target via a mode that it is known to exploit—while sharing a member in common. He uses the LAW matcher to find matches to this pattern, and LAW displays the results.

LAW's display of the matches it found, shown in Figure 5, allows the analyst to see the mappings between the pattern and the matching data in detail. Tables on the left side of the screen describe the pattern-node-to-data-node mappings.[1] The table contains the mappings of the top level nodes in

[1]In the simulated domain, entities are associated only with machine-generated IDs, so the mappings shown in Figure 5 are

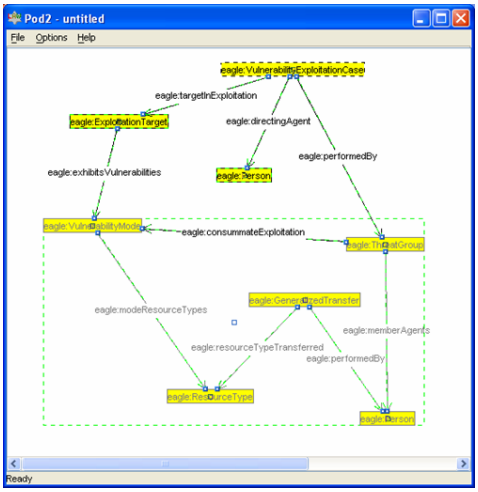

Figure 3: Screenshot of pattern editor

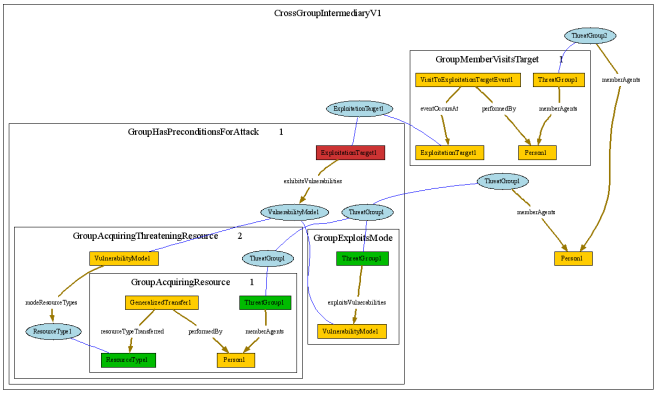

Figure 4: CrossGroupIntermediary, version 1—Pattern representing two groups cooperating through an intermediary who is a known member of both groups

the pattern, and contains buttons for each subgraph that expand into full mapping display tables for them if pressed. On the right of each table is a graphical summary of the match that shows, via color-coding, which pattern nodes and links were matched in the data, and which were missing.

For this pattern in this data set, LAW finds 23 matches. These represent a good starting collection of scenarios that merit further investigation. But the analyst feels that this set of results still does not cover all the group-cooperation possibilities he should be investigating. In particular, because information about threat group membership is often sketchy and incomplete, the pattern's requirement that the intermediary be a known member of both threat groups seems overly restrictive. He wonders what kind of results he would get if, instead of requiring known group membership, he used repeated communication with members of the group as a surrogate.

He uses the pattern editor to modify the pattern to include this new condition, and the resulting revised pattern is shown in Figure 6. Now the graph represents a situation where the intermediary is linked to each of the two groups in two ways:

1. Directly, through a membership link, and
2. Indirectly, through a TwoWayCommunicateWGroup subpattern, which specifies that the person initiates two or more communications (e.g., phone calls) with known members of the group

Additionally, he changes the maximum allowable cost on the top-level pattern from 0 to 2. Since each node and link in the top level pattern has a cost of 1 (LAW's default), this effectively makes it so that the two group membership links are optional.

For the new pattern, LAW now returns 40 matches. They are presented to the analyst in order, best match to worst. In this case, that effectively means that the candidate intermediaries with the strongest known ties to the two groups—both known group membership and repeated communication with group members—are presented first, and the ones with the weakest ties are presented last.

Not all of the 40 matches to the pattern will turn out to be cooperating groups preparing an attack—presumably, the majority will not. But they represent a good candidate set for further investigation. This further investigation can move outside the relational data—where the analyst uses the matches as starting points for searches for information in other formats (text, video, discussions with colleagues, etc.). Or it can be a continuation of the pattern cycle, where the analyst revises the pattern again to add more requirements that will narrow the set of returned matches to a smaller and more relevant set.

LAW

The purpose of LAW is to help intelligence analysts find instances in the world of generic scenarios comprised of combinations of events and facts about players involved in those events. This problem requires more than a simple database querying capability because of the sources of ambiguity and uncertainty within the process mentioned above—noisy and incomplete data, etc. To deal with these problems and to meet the additional requirement of understandability, we have chosen a pattern representation based on graphs, and a match metric based on *graph edit distance*.

Pattern Representation: GEM

The graphical pattern language used in LAW is called GEM (for Graph Edit Model). A GEM pattern consists of two parts: a graph, representing a *prototype* of the situation of interest; and a collection of edit distance parameters, representing the allowable *deviations* from the prototype, along with the impact these deviations have on the assessed quality of the match. We describe the former here, and describe the latter below under the Pattern Comparison Metric heading.

not terribly informative. In more realistic test domains we have used, mapped nodes in the data are described by more user-friendly names or textual summaries.

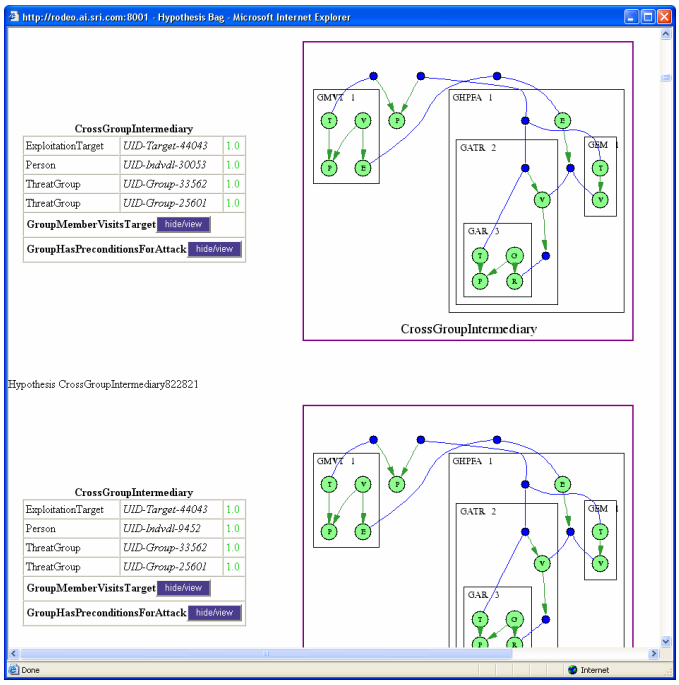

Figure 5: LAW's display of pattern matches

The graph portion of the pattern representation (called the *pattern graph*) is a collection of typed vertices (nodes), and a collection of labeled edges (links) relating the vertices. Each node in the graph is a *generic concept* (Sowa 1984) of a type, or a literal. The types are organized in an ontology. Labels on edges are also typed with classes organized in the ontology. Specific instances can be approximated in the pattern using literals. For example, the person Michael Wolverton can be represented by attaching a PERSON node to a node containing the string "Michael Wolverton" via a NAME relation.[2] Figure 1 shows a simple pattern from the simulated domain described in the next section, representing a member of a threat group visiting a target.

GEM also supports the specification of constraints between nodes. Often these are numeric constraints between attributes representing amounts or dates attached to nodes—e.g., an amount should be greater than $50,000 or an event should occur within two days after a meeting.

Our design goal for the pattern graph representation is to give to the analyst 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 2 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 circular nodes indicate an *interface node*, 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.

[2]This encoding represents *any* person named Michael Wolverton, which is often sufficient for the purposes of pattern matching. If the user needs to identify the person of interest more narrowly, additional qualifiers (e.g., birthdate) can be used.

Pattern Comparison Metric

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. In its simplest form, the graph edit distance between two labeled graphs G_1 and G_2 is the smallest number of editing operations it would take to transform G_1 into G_2. Allowable editing operations are node addition, node deletion, edge addition, edge deletion, node label replacement (i.e., changing the label attached to a node from one term to another), and edge label replacement. This simple model can be extended by adding costs to the various editing operations, perhaps as a function of the labels on nodes or edges, and measuring the edit distance between two graphs as the minimum cost of a sequence of operations that converts one into the other.

LAW uses the more complex model of associating costs with operations. It uses only three of the six aforementioned editing operations: node deletion, edge deletion, and node replacement.[3] LAW supports one other operation not found in other graph edit systems: constraint deletion. Each node, edge, and constraint in a GEM pattern graph has an associated cost for deleting it (see below). For node label replacement, LAW uses the ontological distance between the types of the pattern node and the mapped data node.

The collection of edit distance parameters contained within a GEM pattern specify the allowable deviations from the prototype that will still be considered a valid match, and the cost that various deviations have on the overall quality of the match. These parameters control the calculation of the edit distance between the pattern and the data. They include:

- a *deletion cost* on each node, link, and constraint in the pattern. Each of these can be a number, which roughly

[3]Node addition and edge addition are not relevant in 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.

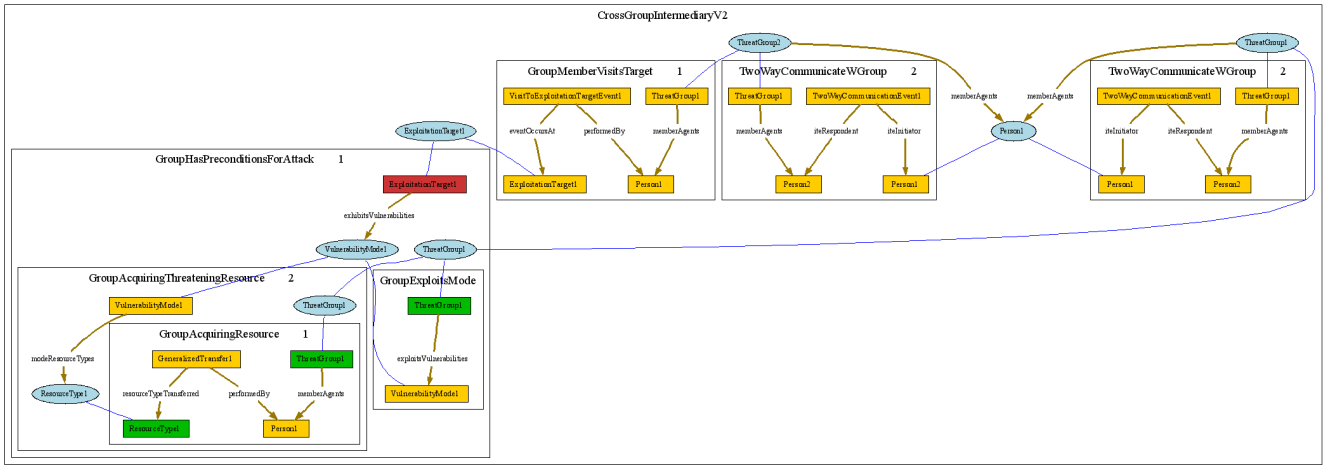

Figure 6: CrossGroupIntermediary, version 2—Revision that allows repeated communication with group members to serve as a surrogate for known group membership

reflects the node's or link's level of importance in the pattern. They can also be set to a symbol representing infinite cost, which indicates that the node or link must be matched by a node or link in the data. The deletion cost on a graph element represents its importance within the pattern, and LAW's display of the pattern shows that importance through color coding—red for an element that has to be matched (high or infinite cost), green for an element that has low cost if unmatched, and yellow otherwise.

- a *maximum total edit distance* for allowable matches. No matches that are above this threshold will be returned, and any partial matches that exceed this threshold will be pruned from the system's search.
- the *maximum number of matches* for the system to return.

LAW's edit distance metric also supports mapping a pattern node of one type to a data node of another type, and associating a cost to this mapping based on the *ontological distance* between the types. See (Wolverton *et al.* 2003) for more detail.

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.

Discussion and Future Work

We have already mentioned several of LAW's features that are specifically geared toward supporting the pattern development cycle. One additional characteristic is LAW's ability to specify and find inexact matches in the data. LAW's edit distance criterion allows an analyst to investigate not just a narrowly-defined set of specific scenarios in the data, but also a broadly-defined *neighborhood* of situations surrounding the prototype graph. This supports the analyst in his exploration of the space of candidate patterns, as well as the rapid focusing on the desired scenario of interest. Other researchers have found that supporting visualization and exploration of the match neighborhood shortens the query refinement cycle (Jones 1998); LAW's inexact match supports this same sort of exploration, albeit in a different way from previous information gathering tools.

One key criterion for any tool supporting the pattern development cycle in intelligence gathering is scalability. The system needs to be able to find matches reasonably quickly even in datasets involving millions of links or more. A num-

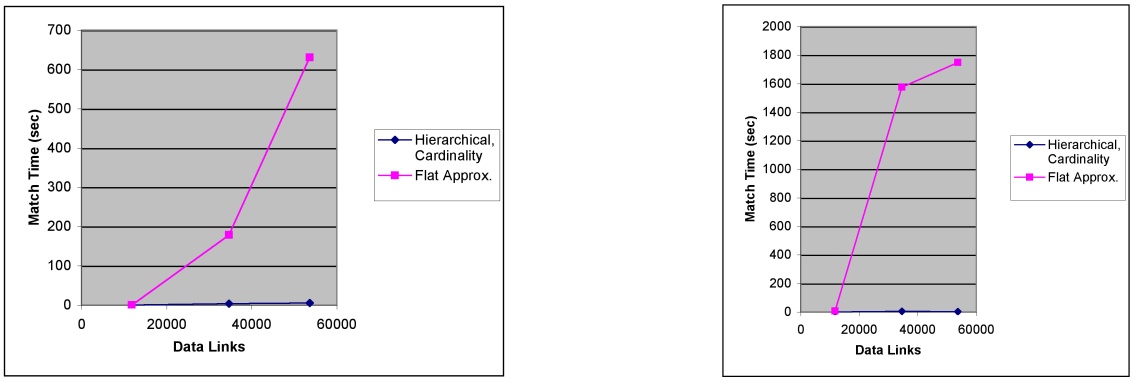

Figure 7: Improved efficiency from hierarchy and cardinality on two different hierarchical patterns—(a) "Group Gets Resources for Mode" and (b) "Hub-and-Spoke"—compared to matching a "flat" approximation of those patterns.

ber of characteristics of LAW are specifically designed to address scalability criteria:

1. The modified A* search LAW uses is designed to find a reasonable set of matches quickly, and then use those solutions to prune much of the search space for the remainder of the search. In addition, as mentioned above, the approach is an anytime algorithm which can halt and return its intermediate matches at any stage in the process.
2. The pattern matcher also utilizes a *search plan* specifying the order in which to investigate pattern elements (nodes, links, and subgraphs). The search plan can be automatically calculated based on statistical analysis of the data, or it can be manually specified by the pattern author if he has particular experience suggesting a specific order of exploration.
3. LAW's data access is implemented on top of a relational database, designed to keep storage and memory use manageable.

Scalability is an ongoing topic of LAW research and development, but the results so far are promising. The higher-order pattern constructs—e.g., hierarchy and cardinality—have given us one or two orders of magnitude speedup over previous versions of LAW. Figure 7 shows some of the results from an experimental analysis of the effect of higher-order constructs on matching time. Additional experiments and results are described in (Wolverton & Thomere 2005). We anticipate further scalability improvements from search plans and other mechanisms still under development.

References

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.

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.

Jones, S. 1998. Graphical query specification and dynamic result previews for a digital library. In *Eleventh Annual Symposium on User Interface Software and Technology (UIST-98)*.

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

Schrag, R. 2006. 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.

Wolverton, M., and Thomere, J. 2005. The role of higher-order constructs in the inexact matching of semantic graphs. In *Proceedings of the AAAI Workshop on Link Analysis*.

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.