Issues in Algorithm Characterization for Link Analysis

Michael Wolverton, Ian Harrison, and **David Martin**
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
wolverton@ai.sri.com

Abstract

To meet the intelligence community's need for link analysis tools that work together, researchers are currently investigating ways of building workflows of these tools using an intelligent system architecture. A key challenge in building a dynamic link analysis workflow environment is representing the behavior of the individual link analysis algorithms being composed. In this paper, we outline techniques for modeling algorithms that allow a system architecture to reason about their behavior and performance, individually and in combination. The algorithm characterization model we propose is based on a layered approach, where the layers range from high-level qualitative descriptions of algorithms to detailed statistical descriptions of their effect on the data.

Recent research and development in technology for intelligence analysis has produced a large number of tools, each of which addresses some aspect of the *link analysis problem*—the challenge of finding events, entities, and connections of interest in large relational data sets. Software developed in recent projects perform many diverse functions within link analysis, including detecting pre-defined patterns (Boner 2005; Coffman, Greenblatt, & Marcus 2004; Pioch *et al.* 2004; Wolverton *et al.* 2003), learning these patterns of interest (Holder *et al.* 2005), classifying individuals according to group membership (Adibi & Chalupsky 2005) or level of threat (Macskassy & Provost 2005), resolving aliases for individuals (Davis *et al.* 2005), identifying neighborhoods of interest within the data, and others.

While these tools often perform complementary functions within the overall link analysis space, there has been limited success getting them to work together. One-time integration efforts have been time-consuming to engineer, and lack flexibility. To address this problem, a recent focus of research has been to link these tools together dynamically, through workflows composed by Grid software (Deelman *et al.* 2003), a blackboard system (Corkill 2003), or some other intelligent System Architecture (SA).

One key challenge in building this kind of dynamic link analysis workflow environment is representing the behavior of the individual link analysis algorithms being composed. In this paper, we outline techniques for modeling algorithms that meet the requirements in the domain of link analysis. The algorithm model we propose is characterized by a layered approach, so that different System Architecture candidates can make use of some aspects of the model, even if they are not capable of reasoning about all of them. The layers range from high-level qualitative descriptions of algorithms to detailed statistical descriptions of their effect on the data. Below we outline the challenges in characterizing link analysis algorithms, describe our proposed algorithm characterization approach in more detail, and discuss related research in algorithm characterization and representing capabilities.

The Problem

The ultimate goal of the dynamic workflow approach described above is to build a fully automated, continuously operating, intelligence analysis support system that composes start-of-the-art link analysis algorithms to achieve the end user's goal. A critical requirement for this type of architecture is that it understand the capabilities and behavior of each individual algorithm, so that it can estimate the outcome of executing algorithms and combinations of algorithms, based on characteristics of the data. This entails a language categorizing link analysis algorithms, along with a software catalog service for storing and retrieving algorithm models based on a variety of requirements.

The key requirements of an algorithm characterization language and catalog are that

- The algorithms are *understandable*, in that they are described at a level that the SA software can ingest and reason with. Further, the algorithm characterization should provide information that the SA can use to prune the enormous search space of possible algorithm workflows it must consider.
- The algorithm models are *precise*, in that they allow fine-grained predictions of algorithm performance.
- The algorithm models are *efficient*, so that predicting an algorithm's performance on a given data set generally takes less time than running the algorithm.
- The individual algorithm models are *composable*, so that the SA can accurately predict and reason about combinations of algorithms run in sequence or in parallel.

Layer Name	Contents	Formalism	Where the Knowledge comes From
Capabilities Layer	Qualitative "before/after" descriptions of data, hard resource constraints	Preconditions and postconditions in OWL or related	Hand-coded by algorithm developers
Data Transformation Layer	Statistical "before/after" descriptions of data	Problem $\times$ Data Model $\Longrightarrow$ Data Model	Experimental analysis, theoretical analysis
Accuracy Layer	Statistical description of expected accuracy of algorithm results	Problem $\times$ Data Model $\times$ Accuracy Model $\Longrightarrow$ Accuracy Model	Experimental analysis, theoretical analysis
Performance Layer	Statistical prediction of performance of algorithm	Problem $\times$ Data Model $\times$ Resource Model $\Longrightarrow$ Performance Model	Experimental analysis, theoretical analysis

Table 1: Multilayered algorithm characterization

Critical Technology Barriers

The algorithm characterization problem presents formidable technical challenges. The bulk of algorithm analysis within computer science has looked at algorithms narrowly (focusing largely on one, relatively uninteresting, case—the worst case) and has characterized their performance very imprecisely (using asymptotic complexity). Both of these features make traditional algorithm analysis highly unsuitable for the detailed predictions needed for this problem. While there have been many more focused quantitative and experimental evaluations of algorithm performance in artificial intelligence (AI) and other fields, those evaluations have been widely divergent in terms of the independent and dependent variables measured. What we need, by contrast, is an algorithm characterization approach that is unified (a variety of algorithms described and measured in the same terms), general (in terms of the spectrum of cases for which it can predict performance), and accurate (in terms of the fidelity of the predictions). To our knowledge, there has been no large-scale attempt to describe a wide variety of algorithms in such a common and detailed formalism.

Existing Commercial and Industry Standards

Commercial use of formal algorithm characterization is still extremely limited, and thus there have been very few standardization efforts in this area. The most directly relevant efforts are the Process Specification Language (PSL), OWL for Services (OWL-S), and the Interoperable Knowledge Representation for Intelligence Support (IKRIS) scenario ontology. PSL (Gruninger 2003) is a first-order logic axiomatization of a set of concepts that may be used to describe processes, and is close to becoming an ISO standard. However, it is best viewed as a means of exchanging process-related information between applications that employ disparate representational schemes. (Indeed, this is what motivated its development.) Although many useful things can be done with PSL, it is extremely abstract and would not provide a convenient or optimal representation for the algorithm characterization challenge described above. The same may be said regarding the IKRIS scenario ontology, which, like PSL, is designed to provide a medium of interchange between heterogeneous applications. However, it will very likely be valuable at some point for algorithm characterizations to be mappable into IKRIS, so as to make them more readily shareable between a variety of analysis tools.

OWL-S (Martin *et al.* 2004) is a candidate for the representation of the capabilities layer, given that it already has an emphasis on specifying inputs, outputs, preconditions, and effects of processes. In addition, a number of open-source tools are available for use with OWL-S, and a good deal of work exists related to its use in the selection and composition of services or processes, as described in Section 4.0 below. OWL-S, while not a standard, has been acknowledged by the World Wide Web Consortium as a Member Submission, and is expressed in OWL (McGuinness & van Harmelen 2004), which is a standard. It is important to note that OWL by itself is not expressive enough for full-fledged representation of general expressions about state (such as the preconditions and effects of algorithms). Indeed, OWL-S allows for the use of quoted expressions from more expressive languages, such as the Semantic Web Rules Language (SWRL) (Horrocks *et al.* 2004), for this purpose.

Algorithm Characterization

The problem of designing an algorithm characterization language for link analysis presents a number of research challenges, many of which arise because of the need for flexibility in the algorithm description. The representation must be flexible in order to accommodate multiple SA candidates that reason about algorithms at differing levels of fidelity, and also to accommodate the evolution of SA candidates' reasoning abilities as they are developed over time.

To meet this need of flexibility, we propose to develop a layered approach to algorithm characterization, where each algorithm is represented at multiple levels of fidelity, and where SA candidates can retrieve and reason about algorithms at the representation level(s) they can handle. The layers include

- *Capabilities*, which provides a qualitative description of the algorithm's behavior, along with the hard constraints for running it

- *Data transformation*, which describes how the statistical profile of the data (e.g., the number of nodes and links of each type) changes as the algorithm is run over it
- *Accuracy*, which describes how the accuracy of each node and link type in the data changes as the algorithm is run over it
- *Performance*, which quantitatively describes the performance of algorithms (e.g., time to complete, time to produce some results) given a data set with a particular statistical profile

Table 1 summarizes the algorithm representation layers, each of which is described in more detail below.

Figure 1 shows the layered Algorithm Catalog and its connection with the SA. The SA queries the catalog for algorithms by specifying in the Algorithm Description Language (ADL) the data input, data output, and/or qualitative constraints on algorithm run conditions, along with a description of the problem to be given to the algorithm (Figure 1 represents that description in a Graph Unification Language (GUL)). The Algorithm Catalog retrieves from its store of layered algorithm representations a set of matching algorithms, along with data transformation, performance, and accuracy predictions for each algorithm in the queried run situation. The layered representations in the store contain qualitative descriptions of the preconditions required and postconditions established by running the algorithm, along with quantitative functions that map statistical profiles of the input data into statistical profiles of the output data. The algorithms and predictions are returned to the SA, which uses them to compare algorithms, compose multiple algorithms into more complicated workflows, and initiate the execution of algorithms and workflows that meet the identified requirements.

Capabilities Layer

This layer describes at a qualitative, symbolic level what the algorithm is designed to do, and what the requirements are for running it. The basic idea is to characterize algorithms as conventional productions (Laird, Newell, & Rosenbloom 1987) or planning operators (Wilkins 1988). That is, the algorithms are described in terms of the preconditions that need to be met to run them, and the postconditions that are established by running them.

The exact language and ontology to be used in describing algorithms at this level is undetermined, but several approaches exist for describing agents and algorithms qualitatively to use as starting points. PHERL (The Pattern, Hypothesis, and Evidence Representation Language) (Murray *et al.* 2005) is well suited to represent the types of problems an algorithm is designed to solve. OWL (McGuinness & van Harmelen 2004) is a general logic-based description language that has been used successfully in describing scientific and other processes (Deelman et al., 2003). OWL-S (Martin *et al.* 2005) is an extension of OWL specifically designed for representing capabilities in Web services. And the Open Agent Architecture (OAA) (Cheyer & Martin 2001) features a language that agents use to register their capabilities and a software facilitator for mapping requests into capable agents.

As an example of the Capabilities Layer, consider a group detection algorithm. Abstractly, these types of algorithms take as input a data set with a few persons classified as members of particular groups (training instances), and produce as output a data set in which all persons in the data are classified into the threat groups to which they belong (to the extent that the algorithm can accurately do so). A simple representation of a group detection algorithm in the Capabilities Layer might be to have a precondition data set DS described with the predicate (Trained DS 'MemberAgents) and a postcondition data set described with (Classified DS 'MemberAgents). Other algorithms—for example, event detection algorithms—may then be described with a (Classified DS 'MemberAgents) precondition.

The Capabilities Layer serves two main purposes. The first is to allow candidate SAs the flexibility of reasoning about the behavior of algorithms at a qualitative level. Most AI architectures (upon which the SAs will most likely be built) are based on logic-like predicates, and have not yet been used to compose their rules on the basis of statistical state models. While we expect that most SA developers would wish to extend their architectures to take advantage of higher-fidelity statistical models to reason about the data, this will take time. To make progress toward a scalable architecture in the meantime, it is important to have descriptions of the algorithms that the developing SAs can use from the start.

The second purpose is to provide search control for all SAs, including those that can reason with the other, statistical layers. When faced with the problem of computing an optimal workflow from a large number of individual algorithms, SAs need some way to prune the enormous space of algorithm combinations. While several possible sources of search control are available, one easy way would be to use the preconditions and postconditions in the Capabilities Layer to limit the number of considered sequences—for example, allow event detection algorithms to be ordered only after group detection algorithms.

Data Transformation Layer

While the Capabilities Layer allows reasoning about algorithms at a high level, general algorithm characterization for link analysis ultimately requires algorithm descriptions that allow more detailed modeling and prediction abilities. For example, the System Architecture must be able to distinguish between multiple algorithms of the same type (e.g., multiple group classifiers, multiple event detectors), and it should eventually be able to discover interesting workflows that were not anticipated by the algorithms' designers (e.g., running event detection before group detection). Reasoning about the algorithms effect on the data at a *quantitative* level would allow these requirements to be satisfied with greater precision.

In this Data Transformation layer, each algorithm is described as a function that maps a problem description and a data model into a data model. The problem description will vary depending on the type of algorithm; for example, a

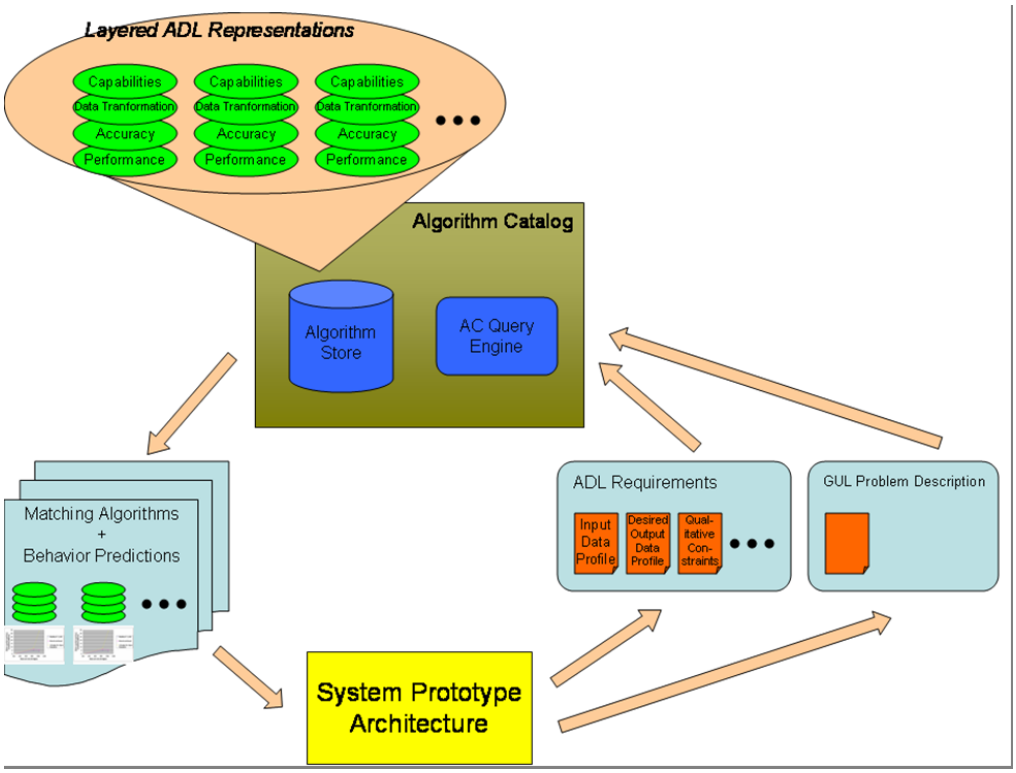

Figure 1: Algorithm Characterization and its role in an automated workflow architecture

problem for a graph matcher could consist of a pattern graph and various parameters specifying the match criteria. A data model is a statistical description of the data set along whatever parameters are determined to be useful in selecting and composing algorithms.[1]

It is possible to view a large variety of link analysis algorithms within this data transformation framework, in which algorithms are characterized by what the data looks like (statistically) after they are run, compared to what it looks like before they are run. Group detection algorithms (Adibi & Chalupsky 2005) will transform the data by increasing the expected number of group membership links. Event detection algorithms will increase the expected number of event nodes, along with links that describe the events. "Good guy/bad guy" algorithms (Davis *et al.* 2005; Macskassy & Provost 2005) will reduce the number of candidate persons in the data graph. "Suspicious neighborhood" algorithms can be viewed as reducing the number of (possibly certain types of) nodes and links in the data.

While the initial models at this layer will be built assuming a static data set, the models will subsequently be modified to account for streaming data. The nature of the streaming data model will vary from algorithm to algorithm. That is, depending on their ability to store and reason over intermediate hypotheses, some algorithms will be able to treat newly incoming data as a separate data set, while others will need to be run repeatedly over the entire growing data set.

Accuracy Layer

Like the Data Transformation Layer, the Accuracy Layer represents the statistical effect the algorithm has on the data. Where the former represents the statistical profile of the data in terms of the number of node and link types it contains, the Accuracy Layer reflects the accuracy of those nodes and links. In this layer, the algorithm is characterized as a function from a (problem description, data model, accuracy model) triple into an accuracy model. An accuracy model is a description of the accuracy of the relations contained in the data set: for all link types L in the data, what is the number (percentage) of false positives and false negatives of that type.

Explicitly modeling the accuracy of data and algorithms is somewhat controversial. Since there is generally no ground

[1] A good first cut for modeling many link analysis algorithms algorithms would be to have a statistical description of the data that consists of (1) the number of nodes of each type and (2) the branching factor per node and link type. Here, (2) is the answer to the question: For each entity type T and link type L, given a node of type T, how many expected links of type L are connected to it? This level of representation implicitly assumes that link and node type distributions are independent—that is, that the existence of link L_1 attached to node T_1 tells nothing about the likelihood of link L_2 also being attached to T_1. For many algorithms, especially pattern matching algorithms, this seems to be a reasonable assumption. However, for other algorithms, especially relational classification algorithms, the independence assumption may be too strong.

truth available for real-world data, it is impossible to measure precisely how accurate it is. Nevertheless, it is critical to include accuracy as part of the algorithm model, because an algorithm characterization that does not include it will be severely limited in what it says about the quality of the algorithm's output. For example, one could model the output of a pattern matching algorithm solely as the number of results it returns, but that is meaningful only to the extent that it implies something about precision and recall. The same thing is true for intermediate results from an algorithm in a larger workflow. If a workflow is going to include a group detection algorithm, for example, the consumer(s) of that algorithm will need to know something about the accuracy of the results. Otherwise it would be reasonable for the SA to conclude that trivial, nonsensical algorithms—for example, include every person in every threat group—are preferable to real classification algorithms.

Performance Layer

This layer predicts the algorithm's efficiency given the data set and problem. The Performance Layer maps a (problem description, data model, resource model) triple into a performance model. The resource model represents the performance characteristics of the hardware available to the algorithm—processor speed, amount of physical and virtual memory, network throughput, database response, and so on. The performance model will represent the predicted time to process the data set, and possibly other measures of performance that we determine are useful for selecting algorithms. For example, one could imagine building a more complicated model of performance for an anytime algorithm, which includes a tradeoff between execution time and the number of (and completeness of) the results produced.

Streaming Data

One of the requirements of the System Architecture approach in general and the Algorithm Characterization portion in particular is that they are able to reason about algorithms' performance over streaming (i.e., dynamically growing) data sets along with static data sets. The quantitative layers of the proposed ADL would be well-suited to handling streaming data with the addition of an explicit model of hypotheses. An algorithm's performance on streaming data is modeled as follows. The algorithm starts with a collection of hypotheses—essentially, partial conclusions drawn from runs on earlier streams of data. As new data arrives, the algorithm is run on just that new data along with its previously produced hypotheses. The algorithm may produce alerts (conclusions strong enough to bring to the attention of a human analyst), may produce more hypotheses, and/or may extend or delete some of its existing hypotheses. These hypotheses are modeled as data—i.e., they are characterized using the same statistical features as regular data sets, and are treated as part of the data input and output of the algorithm. When more data subsequently comes in, the cycle repeats.

Not all algorithms will be able to run on streaming data in this fashion, and that ability or inability will be modeled as preconditions in the capabilities layer. Some algorithms have alternative ways of handling streaming data. For example, during the evaluations conducted by ARDA's EAGLE program, CADRE (Pioch *et al.* 2004) looked at data in incremental temporal windows, only examining the data from a limited temporal interval at any one time. This alternative approach should be easily represented using our algorithm characterization framework as multiple runs on smaller data sets.

Related Work

The problem of algorithm characterization and the related problems of algorithm selection and composition have been investigated for several decades, under various research headings. The idea of using formal logical characterizations of the purposes and effects of programs goes back to Turing and von Neumann, but was developed by Floyd, and in a more formal notation by Hoare, in connection with program verification, as we describe below. Similar ideas have been employed and developed in deductive program synthesis, automatic programming, AI planning, agent-based systems, and Semantic Web services.

Program verification is the use of formal, mathematical techniques to debug software and software specifications. In this discipline, computer programs are regarded as formal mathematical objects whose properties are subject to mathematical proof. In the so-called Floyd-Hoare inductive assertion method of program verification (Floyd 1967; Hoare 1969), certain points in a computer program are annotated with mathematical assertions that are supposed to describe relations that hold between the program variables and the initial input values each time control reaches the annotated point. Another approach to program verification is to transform the program into a mathematical function from the input state to the output state. This approach was initially suggested by McCarthy (McCarthy 1963). An alternative approach is to make programs be objects in a logic (typically tree structures) and then to spell out the semantics of programs explicitly with axioms. Scott-Strachey denotational semantics (Stoy 1977) is an example of this approach to program verification.

Deductive program synthesis (Waldinger & Lee 1969; Green 1969; Manna & Waldinger 1980) is the application of theorem-proving techniques to construct a program that meets a given specification. The totally automatic synthesis of useful programs, which requires a combination of resolution and mathematical-induction theorem proving, is probably still beyond the capabilities of current theorem proving technology, although some striking successes have been achieved in some highly specialized domains, e.g., (Bundy, Smaill, & Wiggins 1990). Techniques originally developed for program synthesis, however, have been applied selectively to less demanding problems, such as the composition of library software to meet specifications (Lowry *et al.* 1994). In this area, there are useful programs whose structure is a simple sequence of subroutine calls. Because repetitive constructs (iteration or recursion) can be hidden in the given subroutines, the problem of intermixing resolution and induction is avoided.

Automatic programming (Rich & Waters 1988), sometimes called knowledge-based system engineering, may be viewed as a broader, more general set of approaches to achieve the same ends as deductive program synthesis.

In AI Planning, the objective is to compose a sequence of actions that accomplishes a given goal, when those actions are executed starting from a given state. The representation of an action usually comprises its preconditions (conditions that must hold before it can be successfully executed) and its effects (conditions that will hold following its execution). The best known instantiation of this basic paradigm is in the STRIPS (STanford Research Institute Problem Solver) language (Fikes & Nilsson 1971). Hierarchical Task Network (HTN) planning (Erol 1995) introduces the idea of refining high-level plans by employing a library of action decomposition methods. However, the representation of both decomposition methods and elementary actions continues to be organized around the fundamental notions of precondition and effect.

The field of agent-based systems (ABS) embraces a large space of challenges, with primary emphases on the autonomy of software agents and the ability of teams of agents to collaborate together with little or no human guidance. Of relevance here is the focus on characterizing and reasoning about agent capabilities. As in AI planning, the common denominator of most approaches is the representation of preconditions and effects, often with additional information about the inputs and outputs of the operations that an agent provides. In BDI systems, such as SRI's Procedural Reasoning System (PRS) (Georgeff, Lansky, & Bessiere 1985), these elements are expressed (partly) in terms of the agent's beliefs, desires, and intentions. At each step, a task is selected for execution from the set of tasks whose preconditions and cues are currently true. Task execution, in turn, can result in updates to beliefs, desires, and intentions.

Agent systems have experimented with a variety of styles of matchmaking, and the use of additional kinds of information, such as quality of service, response time, and other kinds of performance characterization. For example, in SRI's Open Agent Architecture (OAA) (Cheyer & Martin 2001), the basic capability description is a logic programming (LP) predicate structure (which may be partially instantiated), and matchmaking is based on LP-style unification of goals with these predicate structures. In addition, both goals and capabilities declarations may be accompanied by a variety of parameters that modify the behavior of the matchmaking routines.

Most recently, these same problems have been the focus of inquiry in the context of research on Semantic Web Services (SWS). This field aims to enable the automation of the development and use of Web services. Web services, as used in commercial practice, are described using the Web Services Description Language (WSDL) (Christensen *et al.* 2001), which, apart from network addressing and transport protocols, declares little more than the legal syntax of service inputs and outputs (normally using XML Schema). Thus, the first challenge in SWS has been the enrichment of service descriptions, which essentially is the same problem as algorithm characterization. OWL for Services (OWL-S) (Martin *et al.* 2004), the pioneering effort in this field, introduces the expression of preconditions and effects in a Semantic Web-compatible manner, and also relies on the ability to use OWL to construct class hierarchies of services. OWL (McGuinness & van Harmelen 2004), a language developed by DARPA and standardized at the World Wide Web Consortium, is well suited to representing and reasoning about descriptions of classes and their properties. OWL-S also includes a process model—a set of ontology elements (classes and properties) that may be instantiated to obtain a (partial or complete) description of the behavior associated with a service.

The Semantic Web Services Framework (SWSF) (Sem 2005) builds out from OWL-S by including some additional concepts (especially in the area of messaging); employing first-order logic, which is more expressive than OWL; and drawing on the axiomatization of processes embodied in the Process Specification Language (PSL). The Web Services Modeling Ontology (WSMO) (De Bruijn et al., 2005), is an EU-funded effort with many of the same objectives and approaches as OWL-S and SWSF. Two of the most significant differences are as follows. First, WSMO emphasizes the importance of mediators—components that translate between disparate representations—of several different sorts. Second, rather than representing processes per se, WSMO focuses on choreography—the pattern of messages flowing between two interacting processes. SRI has had a leading role in the development of SWS technologies (OWL-S and SWSF in particular).

By and large, algorithm characterization in all of these disciplines has been predominantly concerned with what we here call the capabilities layer of description. Where quantitative descriptions have been used, they have most often been used to solve very specialized problems. Further, quantitative descriptions have rarely taken advantage of probabilistic methods to characterize data transformation and accuracy, or to enable predictions regarding the content and/or structure of generated data, as we have proposed here. Another significant difference proposed here is the combined specification of an algorithm's behavioral characterization with the fine-grained characterization of its data products.

References

Adibi, J., and Chalupsky, H. 2005. Scalable group detection via a mutual information model. In *In Proceedings of the First International Conference on Intelligence Analysis (IA-2005)*.

Boner, C. 2005. Novel, complementary technologies for detecting threat activities within massive amounts of transactional data. In *Proceedings of the International Conference on Intelligence Analysis*.

Bundy, A.; Smaill, A. D.; and Wiggins, G. 1990. The synthesis of logic programs from inductive proofs. In *Computational Logic*. Springer-Verlag. 135–149.

Cheyer, A., and Martin, D. 2001. The open agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems* 4(1):143–148.

Christensen, E.; Curbera, F.; Meredith, G.; and Weerawarana, S. 2001. Web services description language (wsdl) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

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

Corkill, D. D. 2003. Collaborating software: Blackboard and multi-agent systems and the future. In *Proceedings of the International Lisp Conference*.

Davis, J.; Dutra, I.; Page, D.; and Costa, V. S. 2005. Establishing identity equivalence in multi-relational domains. In *Proceedings of the International Conference on Intelligence Analysis (IA-05)*.

Deelman, E.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Vahi, K.; Blackburn, K.; Lazzarini, A.; Arbree, A.; Cavanaugh, R.; and Koranda, S. 2003. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing* 1(1):25–39.

Erol, K. 1995. Hierarchical task network planning: Formalization, analysis and implementation. Technical report, University of Maryland at College Park Dept. of Computer Science.

Fikes, R. O., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. Technical Report 43r, AI Center, SRI International.

Floyd, R. 1967. Assigning meanings to programs. In *Mathematical Aspects of Computer Science: Proceedings of Symposia in Applied Mathematics*, 19–32. American Mathematical Society.

Georgeff, M.; Lansky, A.; and Bessiere, P. 1985. A procedural logic. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*. Morgan Kaufmann.

Green, C. C. 1969. Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 219–239.

Gruninger, M. 2003. A guide to the ontology of the process specification language. In *Handbook on Ontologies in Information Systems*. Springer-Verlag.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Communications of the ACM* 12(10):576–580, 583.

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

Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; and Dean, M. 2004. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, at http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: an architecture for general intelligence. *Artificial Intelligence* 33:1–64.

Lowry, M.; Philpot, A.; Pressburger, T.; Underwood, I.; Waldinger, R.; and Stickel, M. 1994. Amphion: Automatic programming for the naif toolkit. *NASA Science Information Systems Newsletter* (31):22–25.

Macskassy, S. A., and Provost, F. 2005. Suspicion scoring based on guilt-by-association, collective inference, and focused data access. In *Proceedings of the NAACSOS Conference*.

Manna, Z., and Waldinger, R. 1980. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems* 2:90–121.

Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T.; Sirin, E.; Srinivasan, N.; and Sycara, K. 2004. Owl-s: Semantic markup for web services. W3C Member Submission 22 November 2004, at http://www.w3.org/Submission/2004/07/.

Martin, D.; Paolucci, M.; McIlraith, S.; et al. 2005. Bringing semantics to web services: The owl-s approach. In *Semantic Web Services and Web Process Composition: First International Workshop*, volume 3387, 26. Springer-Verlag.

McCarthy, J. 1963. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*. North-Holland Publishing Company.

McGuinness, D. L., and van Harmelen, F. 2004. Owl web ontology language overview. World Wide Web Consortium (W3C) Recommendation, at http://www.w3.org/TR/owl-features/.

Murray, K.; Harrison, I.; Lowrance, J.; Rodriguez, A.; Thomere, J.; and Wolverton, M. 2005. Pherl: an emerging representation language for patterns, hypotheses, and evidence. In *Proceedings of the AAAI Workshop on Link Analysis*.

Pioch, N. J.; Hunter, D.; White, J. V.; Kao, A.; Bostwick, D.; and Jones, E. K. 2004. Multi-hypothesis abductive reasoning for link discovery. In *Proceedings of KDD-2004*.

Rich, C., and Waters, R. C. 1988. Automatic programming: myths and prospects. *IEEE Computer* 21(8):40–51.

2005. Semantic web services framework, version 1.0. at http://www.daml.org/services/swsf/1.0/.

Stoy, J. E. 1977. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. Cambridge, Massachusetts: The MIT Press.

Waldinger, R., and Lee, R. C. T. 1969. Prow: A step toward automatic program writing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 241–252.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann.

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)*.