AFRL-IF-RS-TR-2006-203
Final Technical Report
June 2006

MACHINE LEARNING FOR THE KNOWLEDGE PLANE

Oregon State University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. Q274

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-203 has been reviewed and is approved for publication

APPROVED: /s/

ALAN J. AKINS
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY Jr., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)* JUN 2006	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* Jun 03 – Dec 05

4. TITLE AND SUBTITLE MACHINE LEARNING FOR THE KNOWLEDGE PLANE	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER F30602-03-2-0191
	5c. PROGRAM ELEMENT NUMBER 62301E
6. AUTHOR(S) Thomas Dietterich, Brandon T. Harvey, Drake Miller, David Jones, Aaron Gray, Alan Fern and Prasad Tadepalli	**5d. PROJECT NUMBER** Q274
	5e. TASK NUMBER KN
	5f. WORK UNIT NUMBER OW

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Oregon State University 15th and Jefferson Corvallis OR 97331	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA 3701 N. Fairfax Dr Arlington VA 22203-1714 AFRL/IFGA 525 Brooks Rd Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** **11. SPONSORING/MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2006-203

12. DISTRIBUTION AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA 06-413

13. SUPPLEMENTARY NOTES

AFRL Project Engineer: Alan Akins, IFGA, Alan.Akins@rl.af.mil

14. ABSTRACT

In 2003, Dave Clark (MIT) proposed a new abstract internet layer called the Knowledge Plane (KP) whose purpose is to support network applications—such as general fault management, self configuration, and performance management—that require distributed, self-aware cognitive processing. The heart of this idea is to combine techniques from machine learning with new architectural concepts in networking to make the internet self-aware and self-managing. This will require revolutionary advances in both statistical learning methods and network protocol design and implementation. We performed pilot work on the machine learning portion of the Knowledge Plane. This consisted of three components: (a) we wrote a document formulating the various learning tasks that will arise within the Knowledge Plane and within KP applications, (b) we worked with networking researchers to design several KP scenarios for purposes of developing a new DARPA-funded KP program, and (c) we participated in DARPA working meetings and planning workshops to help finalize the proposed KP program.

15. SUBJECT TERMS

Knowledge Plane, cognitive processing, machine learning, self-aware networking, self-managing networking

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Alan Akins
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UL	66	**19b. TELEPONE NUMBER (*Include area code*)**

Phase 2 Abstract

Most machine learning research has focused on problems where training data is abundant and learning amounts to searching for statistically justified regularities in the data. When data is extremely limited, perhaps a single training example, such learning strategies are simply not applicable. In such cases, the only hope for robustly learning is to leverage prior knowledge sources in order to draw justified generalizations. This report describes a pilot study conducted at Oregon State University, where we consider such a learning setting in the domain of furniture assembly. In particular, the goal is for our learner to observe a single assembly demonstration and then, by interacting with rich knowledge sources, output an assembly plan that generalizes to new situations. For this problem, example knowledge sources include a physical simulator and configuration-space planner, among others. Such a system could then be used in the field to teach and/or assist others to perform the assembly task. From a scientific perspective, our study highlights capabilities that a general “knowledge based” learner would need to have in order to solve our problem and others like it. From an application perspective, our study provides an architecture and partial implementation of an assembly task learner, highlighting the main technical barriers. In addition, we provide a user study showing the potentially large benefit that such a system would have on assembly time.

Table of Contents

List of Figures

List of Tables

Phase 1:
Machine Learning for the Knowledge Plane: Technology Assessment and Research Scenarios

1. Introduction

In 2003, Dave Clark (MIT) proposed a new abstract internet layer called the Knowledge Plane (KP) whose purpose is to support network applications—such as general fault management, self configuration, and performance management—that require distributed, self-aware cognitive processing. The heart of this idea is to combine techniques from machine learning with new architectural concepts in networking to make the internet self-aware and self-managing. This will require revolutionary advances in both statistical learning methods and network protocol design and implementation.

To explore this idea further and develop it into a DARPA program, several groups of researchers were recruited to perform a seedling study. We were one of those groups. Our contribution to the seedling study was threefold. First, we co-authored a technical report reviewing the current state of the art in machine learning and assessing the appropriateness of machine learning techniques for the Knowledge Plane. Second, we studied the problems currently confronting the internet, both at the edge of the network (where end users operate) and also in the center of the network (where internet service providers operate and interact). We studied methods for diagnosing networking problems. We spent considerable time studying a scenario developed by Jennifer Rexford in which a routing problem is introduced by one ISP when they make a typographical error in their Border Gateway Protocol (BGP) routing tables. Finally, we conducted a detailed study of the internet Domain Name System and the ways in which it can be misconfigured, and we studied methods for applying learning and model-based reasoning to diagnose DNS problems.

2. Methods and Procedures

For our review report, Pat Langley and I studied existing articles in the machine learning literature (several of which we originally wrote) to identify existing work that would be relevant to the Knowledge Plane concept. We also performed thought experiments to develop ideas for how machine learning methods could be useful for the KP.

For our general study of internet problems and diagnosis approaches, we read several papers recommended by other seedling researchers, particularly Jennifer Rexford. These included the following papers:

Labovitz, C., Ahuja, A., Jahanian, F. (1998). Experimental Study of Internet Stability and Wide-Area Backbone Failures. Tech Report CSE-TR-382-98. Department of Electrical Engineering and Computer Science, University of Michigan.

Feamster, N., Winick, J., Rexford, J. (unpublished manuscript) BGP emulation for domain-wide route prediction. Shared with us by J. Rexford.

Williams, B. C., Ingham, M. D., Chung, S. H., Elliott, P. H. (2003). *Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software*, 91(1), 212-237.

Akinci, B., Fischer, M., Levitt, R., Carlson, R. (2000). Formalization and automation of time-space conflict analysis. CIFE Working Paper 59. Center for Integrated Facility Engineering, Stanford University.

Our work on model-based diagnosis for the Domain Name System, we read the following books and papers:

Abbey, J. Mulvaney, M. Ganymede: An Extensible and Customizable Directory Management Framework. *Proceedings of the 12th Systems Administration Conference* (LISA'98). 1998.

Albitz, P., Liu, C. *DNS and BIND*. O'Reilly. April, 2001.

Barham, P., Isaacs, R., Mortier, R., Narayanani, D. Magpie: online modelling and performance-aware systems. *Proceedings of 9th Workshop on Hot Topics in Operating Systems* (HOTOS IX). 2003.

Brownlee, N., Claffy, kc, Nemeth, E. DNS Measurements at a Root Server. Proceedings of the *IEEE 2001 Global Telecommunications Conference*. IEEE, 2001.

Chen, C.S., Tseng, S.S., Liu, C.L., Ou, C.H. Building a DNS Ontology using METHONTOLOGY and Protégé-2000. *Proceedings of the 2002 International Computer Symposium (ICS2002) Workshop on Artificial Intelligence*.

Chen, M., Kiciman, E., Fratkin, E., Brewer, E., and Fox., A. Pinpoint: Problem determination in large, dynamic, Internet services. *Proceedings of International Conference on Dependable Systems and Networks* (IPDS Track), pages 595-604, 2002.

Danzig, P., Obraczka, K., Kumar, A. An Analysis of Wide-Area Name Server Traffic. *Proceedings of the ACM SIGCOMM'92 Conference*. ACM, 1992.

Giap, C. S., Kadobayashi, Y., Yamaguchi, S. Zero Internet Administration Approach: the case of DNS. *Proceedings of the 13th International Conference on Information Networking* (ICOIN'98), January 21-23, 1998.

Harvey, B. *DNS Installation and Configuration*. Tech Report. 2003

Huck, P., Butler, M., Gupta, A., Feng, M. A Self-Configuring and Self-Administering Name System with Dynamic Address Assignment. *ACM Transactions on Internet Technology*, Vol. 2, No. 1, February 2002, Pages 14-46.

Koller, D., Levy, A., and Pfeffer, A. P-CLASSIC: A tractable probabilistic description logic. In *Proceedings AAAI-97*, pages 390–397, 1997.

Langfeldt, N. Troubleshooting DNS. Chapter 6 of *The Concise Guide of DNS and BIND*. QUE, 2001.

Lewis, L. *Managing Computer Networks, A Case-Based Reasoning Approach.* Artech House. 1995

Ogle, D. et al. Canonical Situation Data Format: The Common Base Event. October, 2003.

We then configured a set of four computers to mimic the top levels of the DNS hierarchy and to mimic end-user computers at the edge of the DNS hierarchy. This helped us to understand better how the DNS works. We then performed experiments on this mock-up network in which we injected misconfiguration faults and observed the resulting symptoms in the network.

With the help of the network administrators at Oregon State University, we monitored all outgoing DNS queries from Oregon State University for a week and analyzed this data. We applied various DNS tools to analyze the DNS configuration at Oregon State.

3. Results and Discussion

3.1 Technology Assessment Document

Working with Pat Langley, we wrote and delivered a document entitled “Machine Learning for Cognitive Networks: Technology Assessment and Research Challenges” (attached as Appendix B). This document described the following problem formulations studied in the machine learning literature:

1. **Learning for classification and regression**. The goal here is to predict some value y given some input object x. A related goal is to predict the rest of x given some part of x. A third formulation is to predict the probability P(x) of a given event x.
2. **Learning for planning and acting**. This includes iteratively picking an action a to execute given the observed current state s of the environment and some overall goal. A second formulation involves choosing a sequence $<a_1, a_2, \ldots, a_n>$ of actions given the starting state and the goal. A third formulation is to find some state s that optimizes an objective function $J(s)$.
3. **Learning for interpretation and understanding**. The goal here is to parse a data stream into a tree-structured set of events or activities.

Our paper also reviewed several problem solving tasks that could arise in the area of cognitive networking. These included the following:

1. **Anomaly detection and fault diagnosis.** The central task of the knowledge plane is to detect anomalies and faults. Machine learning methods can be applied to learn what “normal” network configurations, network traffic, etc. look like and thereby detect anomalous network traffic. For fault diagnosis, supervised machine learning could be applied to learn rules mapping from sets of symptoms to possible causes. Alternatively, learning could be applied to learn explanatory or causal models to explain how faults lead to the observed symptoms or anomalies. Finally, diagnosis often requires making active probes or measurements (e.g., injecting traffic into a network to see how it responds). Machine learning could be applied to learn the background probabilities of different faults, the cost of different probes, and the probability that a particular fault gives rise to a particular probe outcome.
2. **Responding to intruders and worms.** Intrusion and worm detection can be divided into two tasks: (a) recognizing known intrusion exploits and worms and

(b) detecting new intrusion methods and new worms. Task (a) is an instance of learning for interpretation: the system must learn to detect the sequence of events corresponding to an intruder or worm, even if they are mixed together with other events intended to mask the intrusion. Task (b) is another instance of anomaly detection.

3. **Network configuration and optimization**. We identified a spectrum of configuration and optimization tasks ranging from simple parameter selection through compatible parameter configuration, topological configuration, and component selection and configuration. Aside from simple parameter selection, there has been little machine learning research on these problems.

The conclusion from this study was that while existing machine learning methods could be of great help in detecting and diagnosing problems, the problem of learning for configuration and optimization is an important direction for future research. Hence, two DARPA programs are conceivable. One program would fund research on adapting existing machine learning methods to problems of anomaly detection, fault diagnosis, and intrusion detection. A second program could fund research on new learning algorithms for configuration and optimization.

The report identified additional research challenges for machine learning arising out of the Knowledge Plane domain. These included the following:

1. **Making machine learning more autonomous**. Existing learning methods rely on careful engineering of the representation, the choice of algorithm, and so forth. The knowledge plane requires a more autonomous learning capability that can operate in real time without continuous knowledge engineering.
2. **Learning online**. Most existing learning methods learn off-line from data collected in advance. The knowledge plane requires learning online from data gathered from the “live” internet.
3. **Learning in changing environments.** Existing learning methods assume that the mapping from inputs to outputs is fixed and not changing over time. However, network conditions in the internet are changing rapidly in response to the growth of the size of the network, the amount and type of traffic in the network, and the active attempts of intruders to elude existing counter-intrusion tools.
4. **Distributed learning**. Existing learning methods are centralized. All data is collected, pooled in a central location, and then analyzed by the learning algorithms. The Knowledge Plane requires distributed learning both because of the sheer size of the network and also because data may be proprietary or private, so it cannot be shared in a central location.
5. **Knowledge-rich learning**. Existing learning methods are knowledge-lean. The Knowledge Plane requires methods that can exploit existing knowledge about internet protocols and management policies rather than having to rediscover these.
6. **Learning declarative models.** A natural direction to explore for analysis and configuration of the internet is model-based programming. These techniques were developed by NASA for the Deep Space One mission, in which a space craft was controlled autonomously by an on-board diagnosis, repair, and scheduling

system. An important direction for future research is to develop methods for learning the kinds of models required for model-based diagnosis and configuration.

3.2 BGP Prepending Typo Scenario

Working with Jennifer Rexford (AT&T) we developed a scenario for diagnosing a BGP prepending typo misconfiguration. The BGP protocol is used to control the high-level routing of packets in the internet. The internet is divided into regions called Autonomous Systems (ASes). Each AS has a number, and a path through a sequence of ASes is represented by a sequence of AS numbers. Each AS uses the BGP protocol to advertise ranges of internet addresses to which it can deliver packets (directly or indirectly). Each advertisement consists of an IP address range and a sequence of AS numbers describing the path along which packets will be sent to reach those IP addresses. Unfortunately, the Border Gateway Protocol does not provide a way to indicate the "cost" of sending packets along an AS path. The only measure of the cost of a path is the number of AS numbers in the path. One way to signal that a path exists but is expensive is to "prepend" an AS number multiple times. For example, Autonomous System number 43 might advertise that it can reach IP address range 55.22.*.* by following the path 43 43 43 43 2332 5591. The four 43s are used to make this path artificially long (and therefore, encourage the packets destined for 55.22.*.* to take a different route). Prepending requires manual editing of a configuration file on the border gateways, and a typographical error can result in paths that cannot reach the indicated addresses. So for example, if "43" were mistyped as "34", the AS path would become 34 34 34 43 2332 5591, which would send the packet to AS 34 which might not be able to reach 55.22.*.*. This results in lost packets and a cascade of potential failures.

To diagnose a prepending error, it is necessary to compare advertised paths with the known connectivity of the internet. For example, if AS 34 is not directly connected to AS 43, then this means the path is bad. There is no central authoritative "map" of internet connectivity. Instead, there are some internet resources, such as routeviews.org, which collect BGP advertisements and analyze them to construct a connectivity map. Consulting routeviews.org for each routing decision is not feasible, so a method is needed for detecting routing problems, identifying cases in which prepending typos may be the cause, and then verifying them at routeviews.org. We developed a scenario that described the sequence of inferences that would need to occur including the sequence of passive and active probes or data measurements that would need to be made.

We drew three conclusions from this study. First, model-based reasoning is an important method for diagnosing routing problems. When diagnosing BGP problems, Dr. Rexford clearly applies a very strong model of how BGP works and how network administrators enforce routing policies. Second, existing model-based methods are not adequate for reasoning about the internet, because they assume that the structure ("anatomy") of the system is fixed during diagnosis, whereas the structure of the internet is unknown and changing rapidly. A third conclusion is that while machine learning could contribute to making diagnosis more efficient, it is not the key technology for advancing the Knowledge Plane. Instead, it seems that model-based diagnosis, suitably extended, will be the key to performing rapid, reliable diagnosis of internet misconfigurations.

3.3 DNS Diagnosis Study

In consultation with Bob Braden and Ted Faber of ISI, we performed a detailed study of approaches for diagnosis DNS misconfigurations. The goal of this study was to assess how well existing methods of diagnosis could be applied to a relatively simple part of the internet protocol suite. The Domain Name System (DNS) is a distributed set of millions of servers that is responsible for mapping symbolic domain names (e.g., www.yahoo.com) into internet protocol (IP) addresses, such as 216.109.118.71.

To study the Domain Name System, we configured a set of four computers in our lab (isolated from the real internet) to implement a version of the top level of the DNS including the root DNS servers, the top-level domains, the second-level domains, and some ordinary end-user machines. We then wrote scripts to inject various kinds of configuration faults into this miniature internet. The configuration faults were based on faults listed in standard DNS textbooks and how-to manuals. For each fault, we identified a set of symptoms that it creates, and we built a fault-symptom table (see Appendix A). One of the questions we sought to answer was whether it would be possible to build a fault-symptom table or whether the relationship between faults and symptoms is so complex and dynamic that a model-based approach is required. Our tentative conclusion is that for diagnosing standard DNS faults, a model-based approach is not required. A model-based approach might still be desirable because (a) models are easier to maintain over time than fault tables and (b) models can give useful explanations to the user about how the fault causes the symptoms, whereas the fault table does not represent this information.

Our conclusion that fault tables might be sufficient is preliminary, because we ran out of time and money before we had finished our experiments. In particular, the following steps still need to be performed to verify our preliminary conclusions:

1. Finish building the “miniature internet”. Our miniature internet does not currently support second-level domains, nor does it include dynamic DNS.
2. Finish writing the “fault injection” scripts to inject all known faults.
3. Build an infrastructure for parsing log files and performing active measurements (probes).
4. Build an automated diagnostic system to diagnose these faults. This system would use the DNS fault table that we have constructed by hand. If it can correctly diagnose all of the faults we inject, then that would support our claim that a simple fault table is sufficient for DNS diagnosis.

While performing this study, we noticed another issue that could be important to explore. When a DNS server answers a query (e.g., “What is the IP address for yahoo.com?”), it provides an IP address (216.109.118.71) and a Time-To-Live (TTL) that specifies how long it is safe to cache this IP address. Typical TTL values are 1-3 days. During this time, if the user needs the IP address of yahoo.com, he can obtain it from the cache (which is typically stored on the user's own computer and also on DNS servers within the campus, firm, or ISP). After the TTL expires, the user's computer must again make a DNS query to determine the IP address. This query typically goes to one of the root name servers (of which there are only a small number, worldwide). We noticed that several internet sites are using very small TTLs (on the order of 5 minutes). We speculate that they are doing this for load-balancing, since they can change the mapping from domain names to IP addresses dynamically every 5 minutes. But a side-effect of this policy is that they are creating a heavy load on the root domain name servers. This could be slowing down all domain name lookups worldwide. It would be interesting to measure this and see whether it is effectively a form of "cost shifting" where these internet sites are shifting the cost of load balancing onto the users of the internet.

4. Conclusions

We review the main conclusions of our seedling study here.

First, we determined that existing machine learning techniques could be directly adapted to address problems in anomaly detection and diagnosis. Further research would be needed to develop new techniques suitable for configuration and optimization of network parameters.

Second, we determined that model-based reasoning methods have potential for supporting the kinds of reasoning required by the Knowledge Plane, particularly in the complex BGP prepending scenario. For such scenarios, it is difficult to imagine a fixed fault table that could predict all of the symptoms that could result from a prepending error.

Third, we reached the preliminary conclusion that model-based reasoning is not required for diagnosing DNS faults. Instead, we believe that a fault table would be sufficient. However, we were not able to complete the implementation of a diagnostic system based on fault tables to verify this conclusion.

Phase 2-Part 1:
Machine Learning for the Knowledge Plane: Technology Assessment and Research Scenarios II

1. Introduction

Since June 2003, we have been carrying out seedling research to assess the potential benefits and technical feasibility of a DARPA-funded effort in cognitive networking. The goal of cognitive networking is to enable the Internet (both the commercial Internet and also mission-centered military networks) to become self-configuring, self-diagnosing, and self-tuning so that the network is more efficient, more reliable, and responds better to unanticipated problems. Our initial research efforts have focused on identifying the relevant ideas and technology within cognitive systems and computer networking research. The conclusions of our initial efforts are the following:

1. The most promising fundamental technology for cognitive networking is model-based monitoring and diagnosis. The central feasibility question for cognitive networking is whether model-based monitoring and diagnosis can be extended to deal with the dynamics of the Internet.

2. An untapped source of power for Internet monitoring and diagnosis is the capability of vast numbers of computing devices to pool their observations to rapidly localize, diagnose, and repair problems. A central feasibility question for cognitive networking is whether there is a way to harness distributed problem reports to perform rapid localization and diagnosis, or whether the cost of managing this information is too great to make it useful.

3. A potential advantage of cognitive networking would be the ability of the Internet to tune itself automatically to meet the missions of different local parts of the Internet. A central question for cognitive networking is whether it is feasible and worthwhile to have sub networks tune their configuration and communication protocols automatically to support mission-specific goals.

In this report, we provide a summary of our recent effort into study the three questions above, bearing on the feasibility of cognitive networking. To focus our study, we have chosen to investigate whether model-based technique can be used to diagnose problems in the current Domain Name System (DNS) in a distributed manner. In particular, we have built an experimental test bed and designed a set of algorithms to test the following hypotheses as proposed in 2003.

1.1 Hypotheses

A. Model-based reasoning can be extended to support monitoring and diagnosis of DNS problems.

B. A distributed system of model-based reasoning agents (diagnostic agent) can be coordinated to support distributed monitoring and diagnosis of DNS problems.

C. Problem reports aggregated from end-user computers can reduce or eliminate the need for diagnostic agents to make active probes during diagnosis.

D. Feedback provided by diagnostic agents can guide the automatic tuning of DNS parameters to meet local mission-specific goals.

1.2 Deliverables

A. DNS ontology: A machine understandable and explicit representation of the conceptual layer that will facilitate a higher-level representation of DNS settings enabling the application of high-level reasoning techniques.

B. DNS configuration model: A representation of the space of legal and illegal configurations of a DNS server.

C. DNS structural model: Topology of a domain name system in terms of interconnected DNS servers and the interactions among them. This model will cover the interactions underlying the intended behavior model.

D. DNS behavioral model: Logical representation of how a DNS server works and evolves over time according to different parameter settings.

E. DNS symptom database: A high level representation of inconsistencies between the observed behavior and the predicted normal behavior of the system

F. A DNS sensor feed that allows the use of feedback provided by multiple endpoints to improve diagnostic speed and accuracy.

G. A collection of DNS performance metrics in terms of associated traffic and cache statistics.

H. A tool for the creation of reconfigurable DNS settings. This tool will provide a collection of DNS scenarios to test our algorithms.

I. A set of distributed diagnosis algorithms. These algorithms will constitute diagnostic agents' inner logic.

2. Methods and Procedures

To validate the above hypotheses, the research team comprised of Prof. Thomas G. Dietterich, Prof. Thinh Nguyen, Brandon Harvey, Drake Miller, David Jones and Aaron Gray read and studied existing articles from various fields of research in computer science. A list of the referenced works is provided in sections 2.1 – 2.5.

2.1 DNS related references:

N. Brownlee, KC Clay, and E. Nemeth. *Proceedings of the IEEE:* DNS Measurements at a root server. *2001 Global Telecommunications Conference*, 2001.

Barr. RFC 1912: Common DNS Operational and Configuration Errors, 1996.

P. Beertema. RFC 1537: Common DNS Data File Configuration Errors, 1993.

R. Bush. RFC 2181: Clarification to the DNS specification, 1997.

P. Mockapetris. RFC 1034: Domain Names - Concepts and Facilities, 1987.

P. Mokapetris. RFC 1034: Domain Names - Implementation and Specification, 1997.

Pappas, V., Xu, Z., Lu, S., Massey, D., Terzis, A., Zhang, Lixia: Impact of Configuration Errors on DNS Robustness. *ACM SIGCOMM 2004*, Portland, OR.

2.2 Configuration Modeling and Decision Processing references

Heckerman, J. S. Breese, and K. Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38:49-57, 1995.

B. C. Williams, M. D. Ingham, S. H. Chung, and P. H. Elliott. Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE: Special issue on Modeling and Design of Embedded Software*, 91(1):212-237, 2003.

2.3 Description logic references

McGuinness, J. R. Wright. An industrial strength description logic-based configurator platform. *IEEE Intelligent Systems*, 13(4):69-77, 1998.

Jena, semantic web framework. http://jena.sourceforge.net

Deborah McGuinness, Franz Baader, Diego Calvanese, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

Shelly Powers. *Practical RDF*. O'Reilly and Associates, 2003.

W3C, Deborah L. McGuinnes, and Frank van Harmelen. Owl web ontology language overview. http://www.w3.org/TR/owl-features/.

Koller, A. Levy, and A. Pfefer. P-classic: A tractable probabilistic description logic. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), 390-397, 1997.

2.4 Agent architecture references

Martin, E. Plaza, and J. A. Rodriguez. An infrastructure for agent-based systems: an inter-agent approach. *International Journal of Intelligent Systems*, 15(3):217-240, 2000.

Martin, E. Plaza, and J. A. Rodriquez-Aguilar. Conversation protocols: Modeling and implementing conversation in agent-based systems. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, pages 249-263. Springer-Verlag: Heidelberg, Germany, 2000.

Applications, technologies, architectures, and protocols for computer communications, pages 319-330, New York, NY, USA, 2004. ACM Press.

2.5 Logging and monitoring references

IBM. Common base events, http://www06.ibm.com/software/tivoli/features/cei/.

3. Hypothesis Testing

Our primary research focus is to provide an infrastructure that enables more conclusive and continued research in the area of Cognitive Networking (COGNET). This infrastructure includes (a) an experimental Domain Name System (DNS) running on our local network, (b) a set of diagnostic agents, each agent residing on a different DNS server, and (c) an ontological representation of DNS structure used by agents for model-based reasoning. Using this infrastructure, our goal is to test the four hypotheses in Section 1.1. We have shown the first two hypotheses to be plausible, and are in excellent position to test the final two hypotheses. These final two hypotheses can be easily tested using the existing groundwork. We now discuss our progress on testing the first two hypotheses.

3.1 Hypothesis 1: Model-based reasoning can be extended to support monitoring and diagnosis of DNS problems.

To test this hypothesis, our investigation focuses on the delegation errors in DNS. Pappas states that 15% of all DNS problems are due to delegation errors, which makes solving this problem particularly useful. A delegation error occurs when a hostname is either added or removed from one of the delegation points in a domain's hierarchy without making subsequent changes in the remainder of the tree. A delegation error can occur from the bottom up and from the top down which is referred to as inconsistent/cyclic delegation and lame delegation, respectively. *Our initial results indicate that by modeling the DNS environment using an ontological representation, these delegation errors can be systematically diagnosed using an inference engine.* In particular, we model the DNS environment using a Web Ontology Language (OWL), and use the Jena reasoning engine to diagnose the delegation errors. Jena is a Java framework for building semantic web applications. It provides a programmatic environment for OWL and a rule-based inference engine. An ontological representation of a DNS environment represents the relationships between different DNS servers and the properties within a DNS server. While Jena's reasoning engine can diagnose certain DNS errors using an ontology built from the information at the local DNS server, it often requires information from other servers for diagnosis of delegation errors. Hence, we have designed a set of diagnostic agents with each agent residing on a different DNS server to exchange information for distributed diagnosis. This architecture will be discussed in Section 3.1.

The model-based reasoning process works as follows. First, the diagnostic agent collects the raw data such as the configuration and zone files from its name server and/or its parents/children/master/slave in the DNS hierarchy. Thus, the agent's communication is currently limited to its neighborhood to reduce the bandwidth overhead. This data collection is done periodically or triggered by an event, e.g. a "unreachable host" error or a "restart" event recorded in the DNS log file. Second, the diagnostic agent processes the raw data into an RDF model of the DNS server configuration. RDF is a high-level knowledge representation of data, and is designed for easy semantic extraction and reasoning. Third, the diagnostic agent invokes Jena's inference engine to validate or infer any inconsistencies/errors between the OWL model of the DNS error and the RDF model of the DNS server configuration (a snapshot of the current configuration). The current OWL model is manually designed to represent different DNS delegation errors. Jena's inference engine generates a report elucidating any problems. Using this report, the diagnostic agent determines whether any problem exists. If there is indeed a problem, the diagnostic agent consults with the symptom-fault database for the cause and remedy of the problem. Finally, the agent performs the suggested remedy. Figure 1 summarizes the model-based diagnosis process.

One of the benefits of this model-based reasoning approach is that errors can be described in an abstract human and machine readable form. In addition, we leverage the reasoning engine of Jena to systematically discover errors or combination of errors that might be missed by experts due to the complexity of the DNS models.

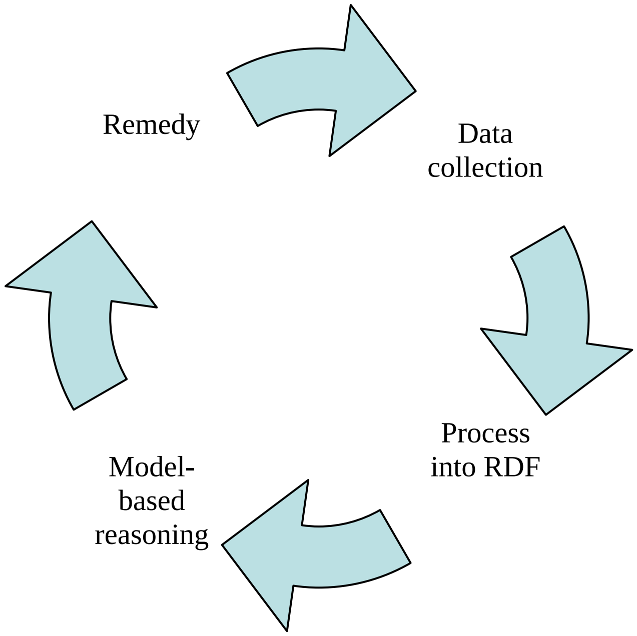


Figure 1 - DNS error diagnosis flow

3.2 Hypothesis 2: A distributed system of model-based reasoning agents can be coordinated to support distributed monitoring and diagnosis of DNS problems.

To test this hypothesis, we have designed a set of agents with the capability to gather local diagnostic information and to communicate with remote agents for information on remote DNS servers. The inter-agent communication consists of passing well-defined message structures using well-established protocols. The information contained in the diagnostic agent's messages consists of high-level models of a server's configuration such as configuration and zone files, and detailed information such as the percentage of failed requests of a particular DNS server during the last hour. This information is then filtered for obvious errors. In particular, the configuration and zone files are passed through a validation process that alerts the diagnostic agent about invalid entries contained in these files. This validation process removes the majority of the human contribution to DNS configuration errors. Next, the diagnostic agent uses Jena's inference engine to capture and diagnose the less obvious errors. *Currently, our diagnostic agents can (a) validate the configuration and zone files and (b) diagnose and correct simple delegation errors involving multiple DNS servers*. Based on our existing infrastructure, we can extend our current models to include many more DNS errors and the corresponding algorithms for correcting or mitigating the problems.

We now describe a simple scenario that our diagnostic agents can automatically coordinate to diagnose and correct the error. We first note that while the DNS errors can be detected by any DNS server, the majority of errors must be corrected by a master DNS server for the particular zone. A master DNS server is an authoritative server that distributes the zone information to all its slave servers. The purpose of the slave servers is to reduce the workload for the master server. Therefore, the majority of DNS errors are discovered by the agents running on slave servers. However, the agents on slave servers cannot change the configuration and zone files of the master server or of the other slave servers. Therefore, when an agent on a slave server detects a potential problem, it must communicate this problem to the agent on the master server. The agent on the master server then makes necessary changes to its master configuration and zone files. It then broadcasts the new configuration and zone files to all its slave servers to correct the problem.

For example, suppose the zone oregonstate.edu has three name servers NS1, NS2, and NS3. NS1 is the master name server, while NS2 and NS3 are slave name servers. Suppose the agent on NS2 detects a configuration error in its zone for eecs.oregonstate.edu and the error is "Lame Server found on ghost.eecs.oregonstate.edu". There are many possible causes for this error, e.g. the removal of the server ghost.eecs.oregonstate.edu. Upon detecting this error, the diagnostic agent on NS2 verifies the name servers assigned for eecs.oregonstate.edu in its own configuration and then forwards the error to the agent on the master name server NS1 for further diagnosis. The agent on NS1 is now responsible for correctly diagnosing the lame server problem with ghost.eecs.oregonstate.edu and making the proper changes, e.g. removing the entry ghost.eecs.oregonstate.edu as a name server. Once the proper changes are made at NS1, the master agent then contacts the agents in the slave servers, NS2 and NS3, to notify them that the changes have been made. Upon receiving the notification, the diagnostic agents on the slave servers inform their DNS slave servers to perform a zone transfer to update their configurations.

In designing the communication between the agents, our aim is to minimize the bandwidth and the computational resource overhead. Therefore, the diagnostic agents employ UDP protocol rather TCP. The UDP protocol reduces the bandwidth overhead as well as the computational resource for transferring data between agents. However, UDP does not provide transmission reliability between agents, i.e. it does not provide automatic retransmission of lost packets. To overcome this disadvantage, the sending agent is responsible for retransmission of the lost packets based on the response from the receiving agent. Furthermore, the diagnostic agent is designed to minimize the number of critical messages that are sent. The communications between diagnostic agents typically include information about their name servers, configurations, and/or explicit instructions for successful diagnosis and correcting problems by another agent.

4. Deliverables

Of the deliverables stated in Section 1.2 the following are complete: DNS Ontology, DNS configuration model, DNS structure model, DNS behavior model, DNS symptom database, DNS sensor feed, collection of DNS performance, reconfiguration tool, and a set of distributed diagnostic algorithms. To summarize, we have either completed or have in place significant foundation in all of the deliverables.

4.1 DNS ontology: A machine understandable and explicit representation of the conceptual layer to facilitate higher-level representation of DNS settings.

A method for creating high-level representations of the DNS settings has been created. We are currently modeling the DNS settings through the use of the Resource Description Framework (RDF) language. RDF allows us to use a statement to represent the settings in much the same way that humans make statements about their observations of the world. For instance we can make the statement that a particular server does not respond to queries. This statement has many implications, one of which could be that there is an error in the configuration settings such as the IP address of the server or the delegation list does not match from one server to the next. Statements similar to the one above are the result of the diagnostic agent first performing routine data collection on the server. The diagnostic agent then performs a minimal amount of pre-processing of the data to extract additional meaning. The high-level data is then used to create an ontology model, which either verifies through inference that the server is operating correctly or determines that errors exist due to the symptoms that exist in the servers' model.

4.2 DNS configuration model: A representation of the space of legal and illegal configurations of a DNS server.

Two XML schema documents are used to describe the valid data contained in DNS configuration and zone files. The information in the XML schemas describes in detail what form the data should take such as the format, range and domain of acceptable values. Through the use of a widely accepted technique of XML validation, simple human errors and malformed configuration data are detected and reported for later use by the diagnostic agent.

The benefit to using the XML schema documents is three-fold. First, XML schemas are a robust and widely used syntax for describing data from virtually any source. Second, XML schemas are relatively easy to understand and learn. Finally, the schema documents are platform-independent and can be extended to describe any type of configuration.

4.3 DNS structural model: Topology of a domain name system in terms of interconnected DNS servers and the interactions among them.

To conduct experimental research in DNS, we constructed an environment similar to the real DNS. This experimental DNS environment is running on five Linux machines, and it allows us to precisely control different parameters, e.g. DNS request rates and injected error rates. Our design is similar to the design used by Wessels and Klaffy in their work on measuring the upper DNS hierarchy.

Similar to the real DNS hierarchy, our DNS hierarchy consists of three fundamental layers: The root servers, the top level domain (TLD) servers, and second level domain (SLD) servers. These are described in more detail as follows:

- **Root Servers:** The root is denoted by a "." and the root servers are used only to direct DNS queries to the proper Top Level Domains. There are 13 root servers, and they are labeled alphabetically from a to m.

- **Top Level Domain Servers:** These servers are sets of servers that categorize the Internet into comprehensible units. Examples of TLDs are “com”, “net”, “org”, “edu”. For each top level domain, e.g. “com” or “net”, there can be a maximum of 13 name servers.

- **Second Level Domain:** These name servers are servers maintained by corporations, organizations and institutions. Examples of SLDs are google.com, nero.net, oregonstate.edu. For each SLD, there can be 13 name servers that are considered authoritative. These domains must be purchased and registered with Internet Corporation for Assigned Names and Numbers (ICANN). Another key feature of the SLDs is that they can contain sub-domains called delegation points. For instance oregonstate.edu has the authority to delegate a sub-domain such as EECS. Once delegated, EECS becomes its own authoritative domain underneath oregonstate.edu and is referred to as eecs.oregonstate.edu.

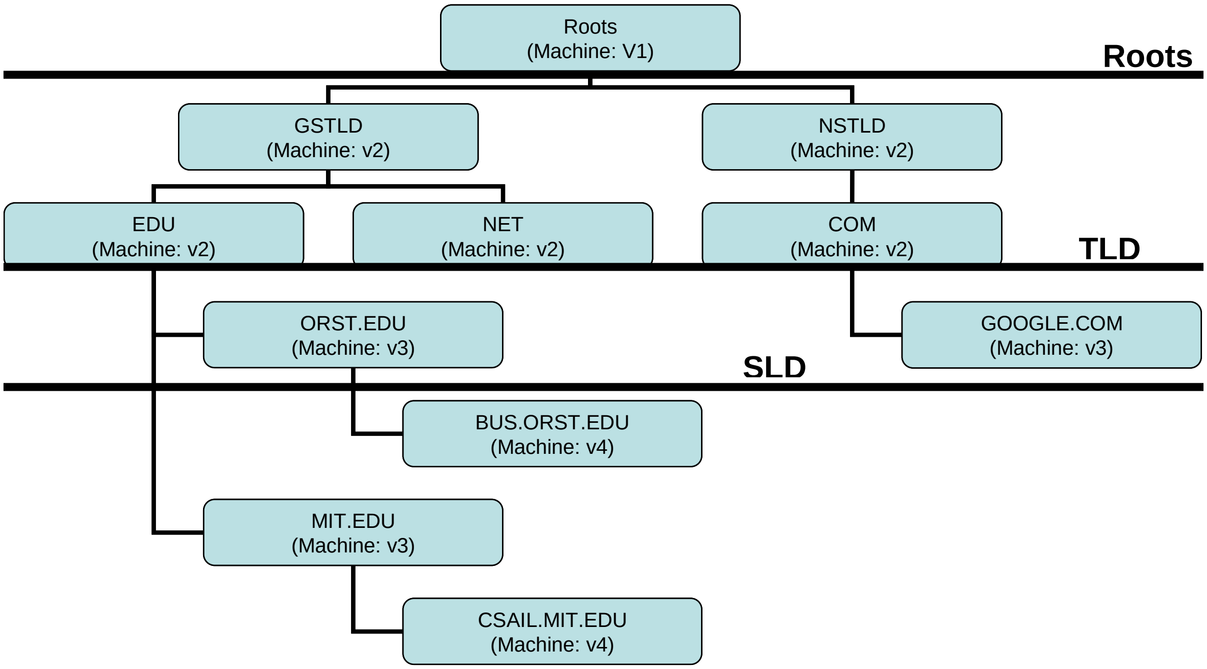


Figure 2 - DNS Hierarchy

For our model DNS, we use five computers to simulate the environment shown in Figure 2. We divide the different layers of the DNS between computers v1, v2, v3, and v4 for simplicity and ease of management. The five computers have the following responsibilities:

- **v1:** Hosts the root name servers a.root-servers.net through m.root-servers.net.
- **v2:** Hosts the top level domains. Currently we have com, net, edu, and org implemented in the model.
- **v3:** Hosts the second level domains. Currently we have orst.edu, ucsb.edu, mit.edu, nero.net, merit.net, and google.com.
- **v4:** Hosts the sub-domains (delegated domains from v3).
- **v5:** Acts as a resolver and metrics server to collect data.

4.4 DNS Behavior Model: Logical representation of how a DNS server works and evolves over time according to different parameter settings.

DNS and other common network services are controlled via text-based configuration files. In the operation of DNS, the configuration files are changed as network requirements change. To change a service, an administrator of that particular service uses a text editor or some other tool to make changes in the associated configuration file(s). This change will subsequently change the operation and overall behavior of the service. DNS service changes only occur when the service is restarted, or for zone files when the serial number is updated.

We believe that by capturing the configuration at the time of the change we can model the behavior of a name server over time. During operation, that is while the service is running, the configuration file can be edited but the configuration will not take effect until the service is restarted. Therefore we can view the operation of DNS and other network services as static. They are only dynamic when viewed over a period of time. If we capture each configuration state at the time the configuration change is made, we will have an accurate behavioral model of the DNS and be able to infer configuration errors with the inference engine.

The reconfigurator algorithm is as follows:

1. Receive state change from Log Adapter.
2. Verify state change.
3. Read configuration files and convert to XML format.
4. Notify Agent of new state.

We have not made much progress toward making a behavior model that can be used to predict and correct and DNS errors in order to improve the DNS performance. However, we have built the framework for future investigations.

4.5 DNS Symptom Database

The DNS symptom database has been created. The symptom database is used by the diagnostic agent to remedy the problems. Please see the Appendix for a complete table.

4.6 Tool for Reconfiguration of DNS Settings

The reconfiguration tool allows us to dynamically change the operations and overall behaviors of the DNS system through settings of the DNS environment. The functions of the reconfiguration tools can be divided into two categories. The first category includes functions that are used to set up the DNS environment. For example, one can use the reconfiguration tool to create or delete the name servers in the DNS hierarchy. One can also use the reconfiguration tool to change the property of a name server such as change the query forwarding method of a name server from non-recursive to recursive. These functions typically involve changing the configuration and zone files of DNS servers. The second category includes functions that assist error creation and statistics collection. These functions aim to measure the performance of the diagnostic agents under different injected error rates and types. For example, to inject a simple delegation error, the tool randomly targets a point in the DNS hierarchy of our test environment and adds an invalid hostname to its delegation point. After the tool makes this change to the target delegation point, it updates the serial number of that zone file so changes can be tracked. It then restarts the name server, which causes the changes to be made. The configuration tool also keeps track of changes for temporal behavior modeling, and also to permit the diagnostic agent to restore the DNS system back to the “last known good” configuration.

4.7 Diagnostic Agent

The distributed diagnostic agent is the core of our solution and envelopes three of the deliverables stated earlier in this document. The deliverables are:

1. A DNS sensor feed that allows the use of feedback provided by multiple endpoints to improve diagnostic speed and accuracy.

2. A collection of DNS performance metrics in terms of associated traffic and cache statistics.

3. A set of distributed diagnosis algorithms. These algorithms will constitute the diagnostic agent’s inner logic.

A graphical depiction of the overall diagnostic agent architecture is shown in the Appendix. The name server agent architecture is divided into three functional components: agent core, reconfigurator, and inference engine. The agent core collects data from local and remote DNS environments to aid the error diagnosis. The reconfigurator provides the agent the ability to capture and make changes to the local configuration. The inference engine uses models generated by the reconfigurator and data collected by the agent core to diagnose possible errors. Also, since an agent is running on each DNS server and it records the behaviors of the DNS server, we are able to learn the behavior of a DNS environment over time, which in turn allows us to answer questions regarding the configuration and operation.

The main function of the agent core is to provide support for the inference engine. This support includes providing the sensor feeds (or probes) and handling of agent-to-agent, agent-to-name server and name server-to-agent communication. The agent core collects information that the inference engine needs to function. It accepts simple pre-defined instructions from the inference engine in order to make the corrective changes to the problems that it discovers.

A secondary function of the agent core is to analyze the name server log files using the Log Adapter. The log adapter is based on the Common Based Event (CBE), part of IBM's Common Event Infrastructure. CBE provides a standard that is platform-independent and can be extended to work with any type of software, including but not limited to DNS Bind. The CBE standard provides a way to categorize and prioritize events logged at the name server. While DNS currently has a category and priority for each event logged, using a standard such as CBE allows us to build a robust and extendable agent that is more capable of tackling problems outside of DNS. This reflects our goal of not limiting our agent to the diagnosis of DNS based configuration problems, but eventually being able to diagnose other network services as well. The log adapter has two specific tasks. The first is to transform new log entries into CBE format and store the entries into a database for archiving. The second task of the log adapter is to filter events based on priority. By setting a high priority, the agent can quickly react to log records that require immediate attention.

The purpose of the inference engine is to provide the agent the ability to infer configuration errors and to monitor the behavior. Since DNS as well as other network services are configuration based systems, we use the Resource Description Framework (RDF) to model the configuration and the Ontology Web Language (OWL) to model the errors. Errors are inferred using the OWL error model and the RDF model of the configuration. The inference engine receives notification from the agent that there has been a configuration change and new data is available from the DNS Reconfigurator. The inference engine then reads in the new data and creates a model of the current DNS configuration. The inference engine then takes the error model and the newly created configuration model to create an inference model. With the inference model, one can ask questions about the properties or values containing in the model. These questions can be general questions such as is there any error in the configuration model? Or, they can be specific questions such as whether a particular error exists. The agent can initiate these questions at a regular interval, at startup, or whenever an error is suspected to be occurring.

5. Conclusions

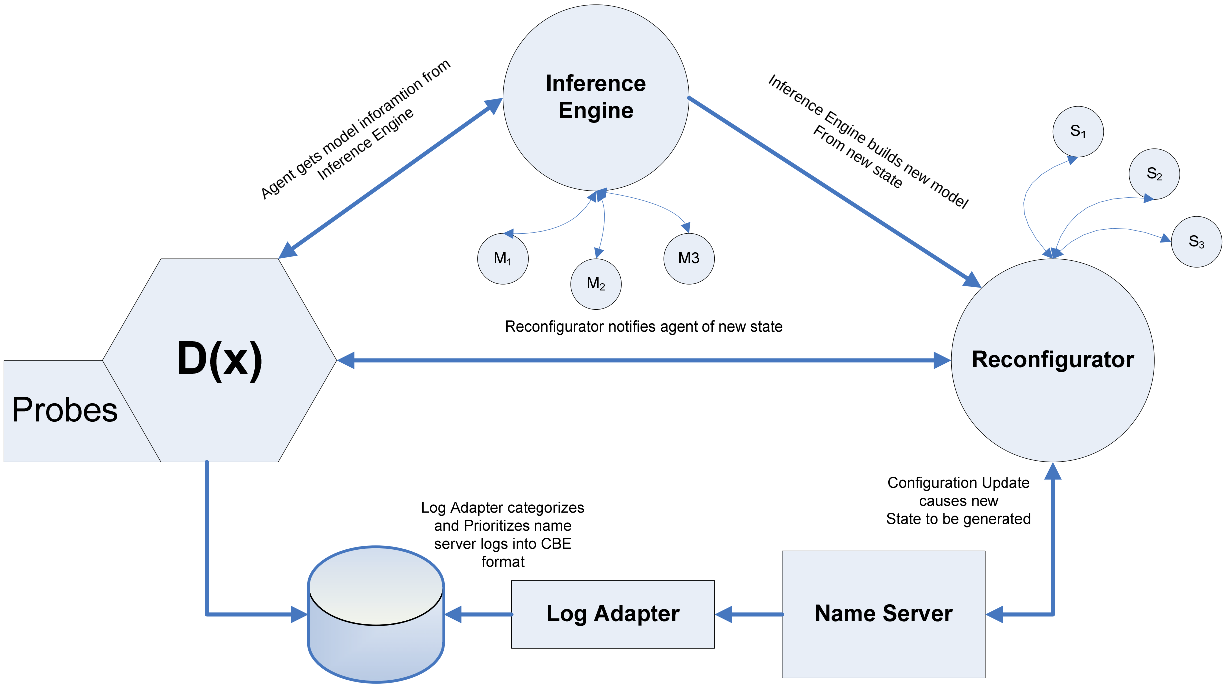
During the course of this seedling study we have made good progress in laying a sound foundation for future research in the field of Cognitive Networking. Through this study we have only begun to realize the potential model-based diagnosis for increasing the reliability and efficiency of a large distributed networked system like DNS.

We have built a DNS structure similar to that used in real-world environments. We have created and deployed diagnostic agents capable of monitoring and collecting applicable data. Finally, we have created a distributed framework for model-based diagnosis, which allows us to diagnose errors in a high-level and scalable manner.

While our research focus has been on diagnosing and correcting DNS delegation errors, we believe an extended diagnosis system based on our framework will be able to correct other DNS errors as well. Furthermore, it is very plausible that one can design a feature-rich software suite based on our model-based reasoning framework that can diagnose and correct errors in other distributed network services.

6. Reference Information

6.1 Diagnostic Agent



Inference Engine
Agent gets model inforamtion from Inference Engine
Inference Engine builds new model From new state
S_1
S_2
S_3
M_1
M_2
M3
D(x)
Probes
Reconfigurator notifies agent of new state
Reconfigurator
Log Adapter categorizes and Prioritizes name server logs into CBE format
Configuration Update causes new State to be generated
Log Adapter
Name Server

Figure 3. - Diagnostic Agent

6.2 Symptom – Fault Table

Table 1: Fault Table

Symptom ID	Symptom Description
s1	Loss of Network Connectivity
s2	Response from Unexpected Source
s3	Recursion Bugs
s4	Client unsure on handleing of NS record in Authority Section
s5	No answer to query
s6	Client calls on server to many times
s7	Nave Server is infected with bogus cache data
s8	A server refers to itself in the authority section
s9	Cache Leaks
s10	Remote Names Can't Be Looked Up
s11	Name Error Bugs
s12	Lookups Take a Long Time
s13	Wrong or Inconsistent Answer
s14	Slave name server data does note change when change is made
s15	Name Server Keeps Loading Old Data
s16	Is invalid proceeding anyway
s17	Slave Server Can't Load Zone Data
s18	Internet services refused
s19	Host fails authentication checks
s20	Inconsistant or Missing Bad Data
s21	Lame-server reported
s22	Name Server fails to load
s23	Name server reports “Too Many Open Files”

Fault ID	Fault Description
f1	Forgot to Increment Serial Number
f2	Forgot to Reload Primary Master Server after changes are made
f3	Corrupt db.cache data
f4	Ignored Referral
f5	To Many Referals
f6	Malicious Server
f7	Zero Answer
f8	Added Name to Database File, but Forgot to Add PTR Record
f9	Incorrect db.cache file
f10	db.cache size set to small
f11	Server does not do negative caching
f12	Syntax error in zone data file on master
f13	Incorrect IP address for master on Slave zone data file
f14	Syntax error in configuration file or zone data file
f15	Missing Dot at the End of a Domain Name in a Zone Data file
f16	Missing root.hints or db.root data
f17	Missing Subdomain Delegation
f18	TTL exeeded
f19	Syntax error in resolv.conf
f20	Incorrect labels in DNS name
f21	Incorrect SOA format
f22	Incorrect Glue records
f23	Retry Interval is to low
f24	Incorrect address in query list – allow-query { address_match_list }
f25	Incorrect configuration named.conf listen-on { ip_address }
f26	PTR record points to CNAME
f27	Expire time exceeded
f28	Loss of Network Connectivity

Phase 2-Part 2:
Learning Generalized Task Knowledge from Demonstration and Question Answering

1. Introduction

Most humans have experienced the difficulties that can arise when using paper-based instructions to complete an assembly task. Most of these difficulties could be avoided if only the instructions “truly understood” the assembly task, could monitor the user, and proactively engage the user. The high-level application goal of this project is to study the use of machine learning techniques for developing such intelligent, proactive instructions.

We imagine a scenario where technicians in the field carry wearable “instruction goggles”, rather than paper instruction manuals. The goggles would contain information about thousands of possible tasks and when worn could guide the user through a particular task via an augmented reality display. For assembly tasks, the augmented reality interface might highlight the next part to be attached to the current sub-assembly along with its destination and tools required. In addition, the interface could proactively monitor the progress of the user and interrupt the user if a mistake is about to be made.

It is already possible to develop such systems for small, relatively simple tasks in closed worlds. However, this comes at a significant and painstaking development cost. Scaling up to thousands of assembly tasks and less constrained environments would require an enormous and impractical development effort. An alternative is to develop machine-learning methods that can acquire the necessary task knowledge automatically by observing demonstrations of each task.

Existing machine learning techniques are typically based on searching for statistically justified patterns in data. For complex learning tasks, such as our assembly example, these techniques would require thousands of training demonstrations, making them impractical. Thus, new, less data hungry, learning algorithms are needed for such tasks. One approach toward this goal is to develop algorithms that can leverage a variety of knowledge sources about the learning domain in order to decrease the training data requirements. For example, useful knowledge sources for learning assembly knowledge might include physics-based and geometric reasoning engines, configuration-space planners, among many others. Learning in the presence of such knowledge sources is a relatively untouched area of machine learning, but is perhaps one of the most important directions toward building truly integrated intelligent systems that can learn.

Below we outline an initial investigation where we develop and partially implemented an architecture for learning furniture assembly tasks from single demonstrations. At an application level, our work highlights some of the main challenges that are left in developing such a system. At a scientific level, our work provides a concrete example of a learning domain where rich knowledge sources are necessary and highlights some of general machine learning issues.

The remainder of this report proceeds as follows. First, we introduce the assembly domain used in our pilot study. Next we give a high-level overview of our learning system's architecture. This will be followed by a more detailed treatment of each system component. Next, we summarize our results from a user study that suggest the utility of having an assembly-task assistant. Finally, we conclude with a summary of the issues raised by our study.

2. Pilot Study Domain

We chose to study furniture assembly as a domain for learning task knowledge from demonstrations. For this purpose, we selected "Z-Line Designers 2 Drawer Deluxe Cherry Vertical File" shown in Figure 4 as our test piece of furniture.

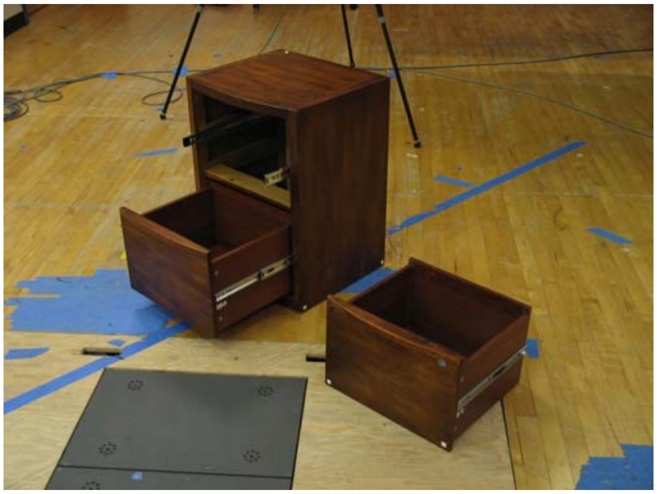
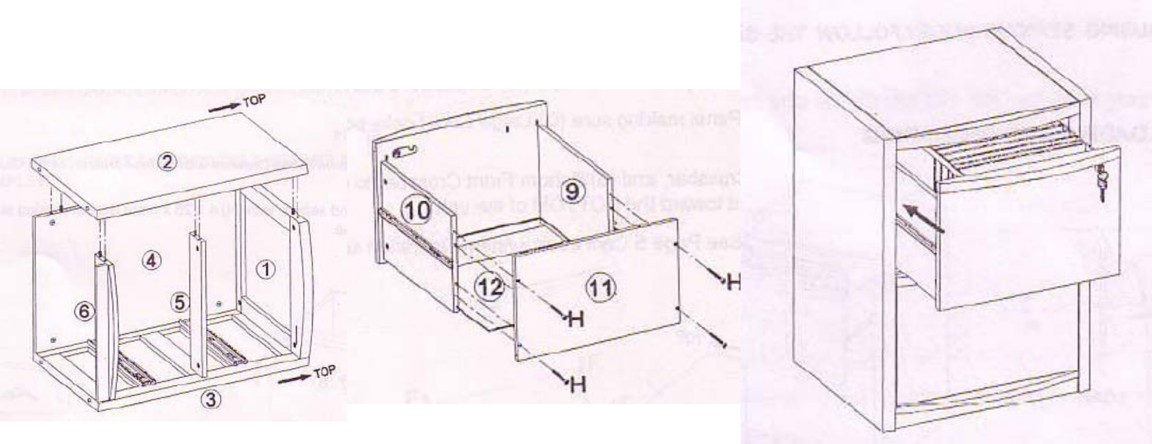


Figure 4. - The Z-Line 2 Drawer Deluxe Cherry Vertical File

The assembly instructions for the Z-Line file can be divided into three parts as depicted in Figure 5a-c. First, the main frame of the file must be constructed. Next, two nearly identical drawers must be constructed. Finally, the drawers are inserted into the file. While this is a relatively simple piece of furniture to assemble, there are many opportunities to make mistakes that lead to dead-ends. For example, if one attaches the four sides of a drawer together before inserting the drawer bottom, it becomes impossible to insert the bottom. Surprisingly, as our user studies show, such seemingly obvious errors are common in practice.

It is interesting to note that although the instruction manual specifies a single sequence of assembly steps, there are many other possible assembly sequences that work equally well.

A human will recognize such generalized task knowledge after observing just a single assembly sequence. We would also like this to be the case for our learner, as an intelligent assembly tutor should not interrupt the user if they depart from the paper instructions, but are otherwise correct. Clearly no system based on pure statistics will be able to robustly infer alternative assembly sequences from a single demonstration. Such alternatives can only be inferred by reasoning about the observed sequence using rich knowledge sources.



(a) (b) (c)

Figure 5. - (a) Depiction of assembly of main frame. (b) Depiction of assembly of drawer. (c) Final step of placing drawer in main frame.

The primary objective of our system is to observe a demonstration of the Z-Line assembly, using motion-capture technology as a “vision system”, and from that learn a generalized plan for assembling the file. Ideally we would like the generalized plan to encode all legal assembly sequences, rather than just the one observed. In addition, we would like the generalized plan to encode a notion of difficulty, so that it can determine whether one assembly sequence is physically easier than another. The ability to judge the difficulty of an assembly sequence, in addition to the correctness, is important for the system to offer the best possible guidance to a user.

3. System Overview

Figure 6 shows the high-level end-to-end architecture of our proposed learning system. These components are described in further detail in the next section.

3.1 Motion Capture.

The front end of our system is a motion capture system that observes a human performing an assembly and produces a movie of orientations and positions of each part involved in the assembly.

3.2 Annotation.

The raw motion capture data is then annotated by a human with information about the observed subtasks and structural relationships in key frames. Eventually the goal is for this annotation to be acquired via a mix of automated analysis (perhaps the result of learning) and speech-recognition technology. Automating this process was beyond the scope of our study.

3.3 Learning Component.

The learning component takes as input the annotated motion capture data along with three knowledge sources. First, there is an interface to a physics simulator that allows for physical queries to be answered via simulation—e.g. “Is the current assembly stable?”. Second, there is an interface to a C-space planner that allows for queries to be answered about the feasibility of physical actions subject to topological constraints—e.g. “Is there any way to insert part A into slot B?”. Third, the learner is provided with geometric models of all the parts involved in the assembly. An eventual goal is for these part models to be acquired, or refined, via automated learning techniques. However, this again was beyond the scope of this study.

The learner analyzes the annotated motion capture data and via a series of queries to the physical simulator and C-space planner outputs a generalized task model for assembling the Z-line. This task model encodes all of the hypothesized legal assembly sequences along with a measure of the difficulty of each assembly step.

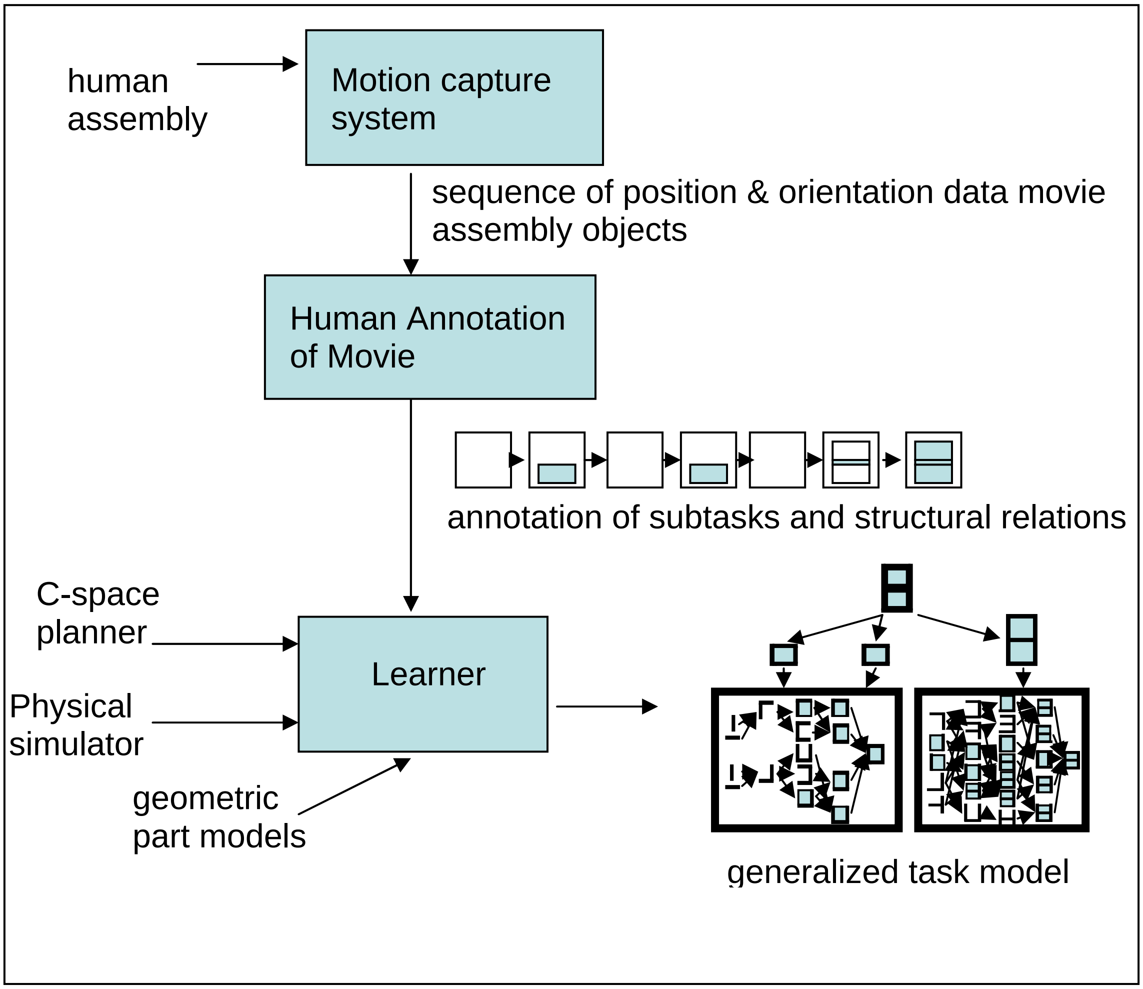

human assembly
Motion capture system
sequence of position & orientation data movie assembly objects
Human Annotation of Movie
annotation of subtasks and structural relations
C-space planner
Physical simulator
geometric part models
Learner
generalized task model

Figure 6. - System architecture of the assembly task learner.

It is important to note that the notion of learning here is much different than the usual one in machine learning. Here we view learning as the process of actively engaging a set of knowledge sources in order to construct a consistent and well-justified hypothesis from observed data. This view is depicted in Figure 7 where the learner is shown to have a two way interface to each knowledge source, allowing it to pose queries and receive responses.

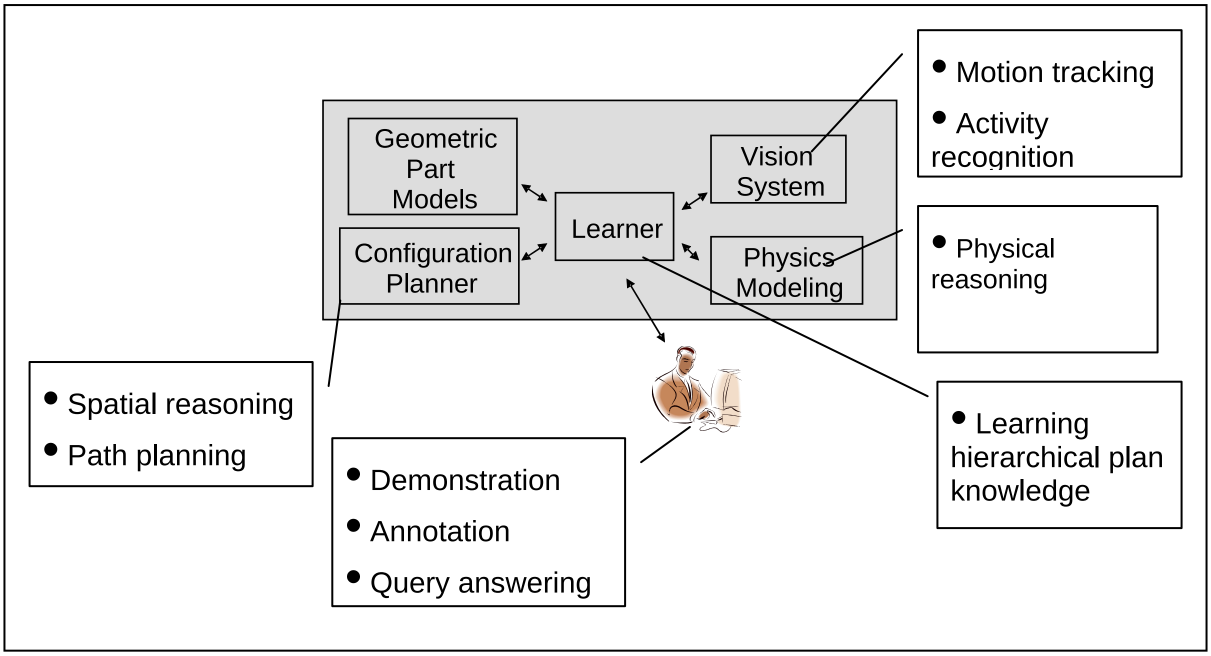


Figure 7. - Depiction of knowledge sources used by learner.

4. System Details

Below we provide more details about the various components and functions performed by our system.

4.1 Motion Capture

Our assembly demonstrations of the Z-line file were conducted in the Oregon State University motion-capture (mocap) facility. A mocap system can be viewed as a simplified computer vision system for tracking objects. The primary simplification is the use of markers on each object to be tracked, which removes most of the difficulties involved with tracking from raw video.

On each part of the Z-line file, we placed three markers A, B, C and chose a position on the part (e.g., one of the corners) to be the origin of the local coordinate system of the part. We measured the displacement of marker A from that origin and recorded that information for each part. Note that we did not attempt to track part connectors (e.g. dowels and screws) or tools in our experiments.

In the motion capture system, the markers all look the same. Whenever a new marker became visible, we manually told the mocap system the id number of that marker.

The mocap system then was able to track that marker until it was occluded by some other part (or by the person, etc.). After completing the mocap for an assembly sequence, an interactive interface was used to clean-up any tracking errors made by the system. It took us approximately 4 hours to manually clean 2 minutes of video. After cleaning, we were left with a movie that gives the 3D (x,y,z) locations of each marker at each time step. This data is then processed to compute the 6 degrees-of-freedom pose of each object in the global coordinate system. Figure 8a and b show a number of the Z-line parts laid out in the mocap room along with an image showing the calculated 3D coordinates of the three markers for each part (shown as triangles).

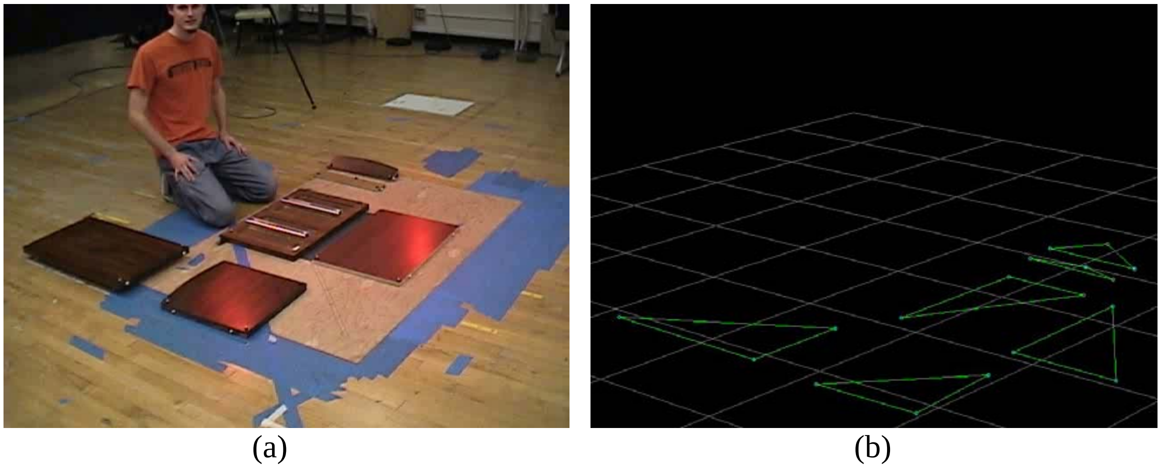


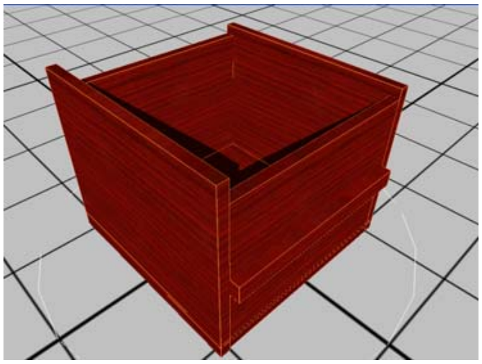
Figure 8. - (a) Cabinet parts laid out at the start of a motion capture session. (b) A depiction of the data recorded by the motion capture system. Each part has three markers, which are shown as the vertices of the triangles in the figure.

After arriving at the final position/orientation movie, we annotated key frames with semantic information about the task and structural relations among parts. In particular, we annotated each frame where a step was completed (e.g. attaching one piece to another) as the “final configuration” of that step. The annotation included information about all of the structural relations among parts. In addition, if the step completed a subgoal (e.g. completing a drawer assembly), we annotate that subgoal. Figure 9a shows an example annotation for one of the key frames.

High-Level Structural Representation:

rigid(DrawerLeftSide, DrawerBack)
rigid(DrawerRightSide, DrawerBack)
rigid(DrawerFront, DrawerLeftSide)
rigid(DrawerFront, DrawerRightSide)
trapped(DrawerBottom, DrawerLeftSide)
trapped(DrawerBottom, DrawerBack)
trapped(DrawerBottom, DrawerRightSide)
trapped(DrawerBottom, DrawerFront)

Position and Orientation Data:
(from marker data)



(a) (b)

Figure 9. - (a) The symbolic annotation of a key frame of the motion capture. (b) Depiction of internal model of key motion capture frame. The internal representation is based on the provided part model, rather than the raw motion capture data.

For any key frame, given the structural annotations, the 3D marker coordinates, and the geometric part models, it is relatively straightforward to use a constrained optimization procedure to create an internal model of the location of each part. This model needs to obey the constraints imposed by the structural annotations and try to match the position data provided by the mocap system as closely as possible. Unfortunately, the timeframe of the project did not allow us to implement this optimization procedure, and we simply provided the internal models by hand for the key frames. Figureb shows a depiction of the internal part model for one of our key frames.

4.2 Generating Feasible All-Policies Graph

A critical question is what representation to use for the learned generalized task knowledge. The representation needs to compactly encode sets of possible assembly sequences, of which there can be exponentially many. We considered a number of representations, including the commonly studied partially-ordered plans and hierarchical task networks. However, these representations did not appear compact enough for our purposes. In particular, physical assembly tasks such as the Z-line file, place weak constraints on many of the action orderings. For example, when assembling a drawer, the order in which the sides are joined together is not important, and our representation needs the ability to compactly encode such information.

We finally settled on a representation which we call an all-policies graph. The nodes of an all-policy graph correspond to partial assemblies, and the edges correspond to single assembly steps. In our study we created multiple all-policy graphs, one for each sub-assembly of the Z-line file, which includes the two drawers and the frame. The intention is that for each sub-assembly, any path through the all-policy graph should correspond to a legal assembly sequence.

Our learner infers an all-policies graph using a two step process. First, given the structural annotation of the final mocap frame of a particular sub-assembly, we create a complete state-space graph by backtracking from the final state. Figure 10 shows the complete state space graph for a drawer sub-assembly. Each arc through this graph corresponds to either attaching drawer sides to one another or inserting the drawer bottom into various grooves on the drawer sides.

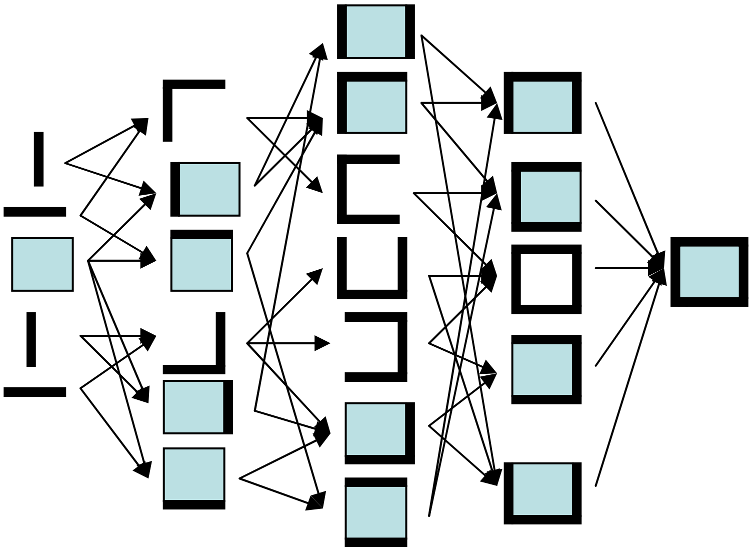


Figure 10. - The complete state-space graph for the drawer assembly. Here the dark edges correspond to the sides of the drawer and the blue area corresponds to the bottom of the drawer. Arrows correspond to the addition of one part to the assembly. Note that some arrows do not correspond to physically possible actions. One of the learner's objectives is to rule out such arcs.

It turns out that some of these arcs in the state-space graph are not physically possible due to topological constraints. For example, as depicted in Figure 11, it is impossible to insert the bottom of the drawer after all four sides of the drawer have been attached. The second step of generating an all-policies graph is to use the C-space planner to determine which arcs are realizable and to prune arcs that are not. We used a monte-carlo-based C-space planner for this purpose. In order to determine whether an arc from state A to state B is possible, we used the planner to determine if there was a topologically valid trajectory for moving the new part into state B. For example, for the lower arc in Figure 11, the C-space planner was set up to find a trajectory for the drawer bottom that placed it in its final location. The planner was unable to find such a plan, so the learner accordingly pruned the arc. After backtracing through the entire state-space graph (pruning when possible) and testing each arc, we end up with a subset of the original arcs that are all topologically possible. The resulting subgraph is taken to be the final all-policies graph Figure 12 shows the all-policies graph for the drawer assemblies (same for both drawers) and for the frame. Also shown is the path taken through the graph in the demonstrated assembly. Clearly the all policies graph has generalized well beyond that single example.

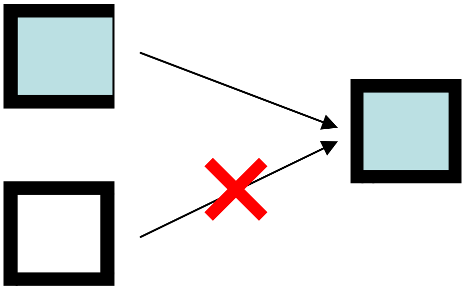


Figure 11. - The lower plan step is infeasible due to the fact that the bottom panel cannot be inserted when all four sides of the drawer are attached. The C-Space planner can be used to determine that the planning step is physically impossible. The top arc corresponds to a physically realizable action. Accordingly the C-Space planner determines that this is the case by computing a plan for attaching the fourth side onto the drawer.

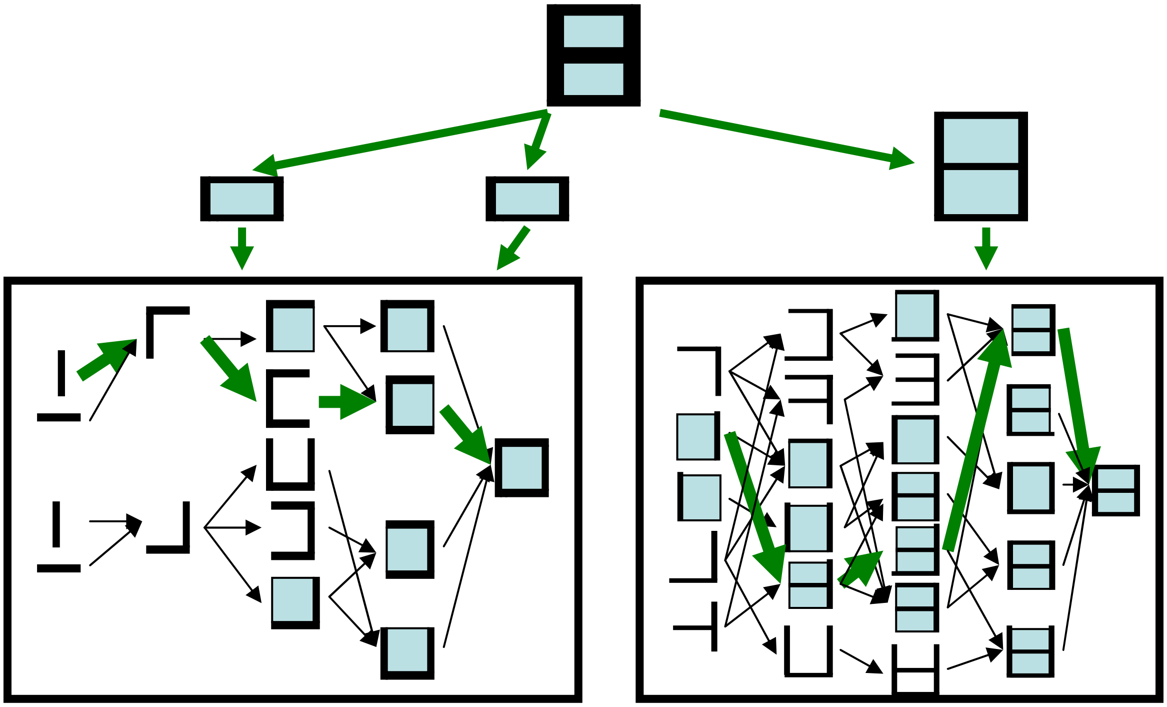


Figure 12. - The final all-policies graph after pruning according topological constraints. The left-bottom plan shows the graph for assembling the drawers. The right-bottom plan shows the graph for assembling the cabinet frame. The green-bold arcs show the observed path from the demonstration.

4.3 Assessing Difficulty of Plan Steps

The above all-policies graph only indicates whether an assembly step is topologically possible or not. It does not encode information about the difficulty of each step. In order to compute such information, the system utilizes the physics simulator. Intuitively we measure the difficulty of an assembly step by roughly how many “hands” are required to perform the step. As a proxy for this measurement, we use the physics simulator to determine the minimum number of parts that must be “frozen in space” in order to make the assembly state resulting from the step stable. This is done by an exhaustive search, where each step of the search freezes some number of the parts and then lets gravity take effect. If any unfrozen part falls due to gravity, then we say that the state was unstable given the current set of frozen parts. We used the Novodex physics simulator in our experiments. Figure 13 shows the all-policies graph with each arc labeled by a weight corresponding to the “number of hands” measure of difficulty.

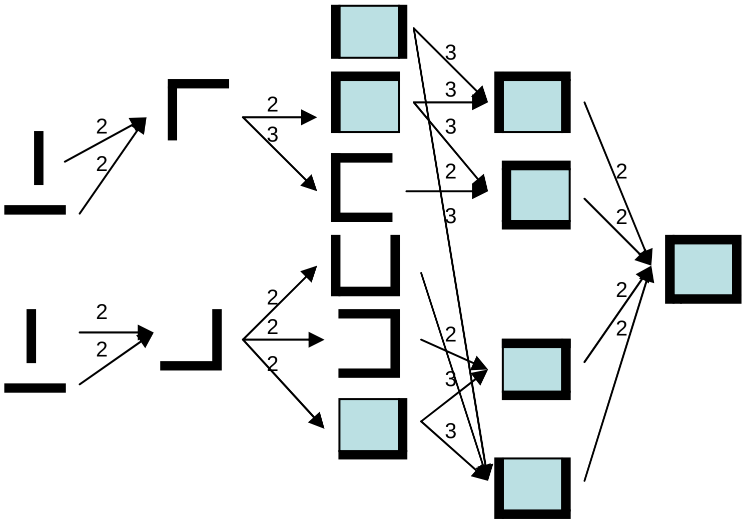


Figure 13. - Weighted all-policies graph for the drawer assembly. The arc weights indicate the difficulty of the planning step and are based on simulating the stability of the partial assemblies.

5. Results of User Study

In order to assess the potential utility of an automated assembly instructor, we conducted a user study with the goal of understanding the typical error modes of humans when performing assembly tasks.

We selected five subjects from the Oregon State University, Computer Science Department and asked them to assemble the Z-line file. The parts were laid out on the table, the cam locks and guide rails were fixed in the appropriate parts, and user was given the Z-line instruction manual. We video taped the assemblies and later analyzed the performance of each participant.

In our analysis we counted two types of errors: 1) detachment errors, occurring when a part is attached and then detached, and 2) deselect errors, occurring when a part was picked up with the intention of attaching it, but then not attached. Deselect errors provide a measure of how useful a human might find a system that highlights the next part to add to the assembly. Detachment errors are quite time consuming and measure the utility of a system that can monitor the user and proactively interrupt them before they make the error. We also separately recorded the total time for assembly and the time spent searching for parts (including reading the manual). **Error! Reference source not found** summarizes the data we collected.

We were most surprised by the fact that the subjects made more errors than we had expected. Three out of the five reached the dead-end state of fixing all four sides of the drawer before inserting the bottom panel (i.e. the arc ruled out in Figure 11), and were forced to backtrack. An automated instructor, using our all-policies graph could have prevented these errors. Another interesting observation is that over 40% of the users' time was spent searching for parts and reading the instruction manual. Determining the proper orientation of parts also consumed a significant amount of the subjects' time. Overall, these results indicate that providing the user with an automated instructor in the form of augmented reality goggles could have a significant impact on assembly performance. The augmented reality display would help the user avoid searching for parts, could show the user the next assembly step, and could demonstrate the proper orientation of each part. The user could also be warned when they are about to take a dead-end action.

Table 2: Summary of user study results.

	Number of Errors		Time (mts:secs)	
	Detaches	Deselects	Total	Search
Average	4.6(31%)	3.8(25%)	19:49	8:20(43.2%)
S.D.	3.8	2.4	6:03	2:32(6.41%)

6. Summary and Conclusions

Our pilot project has developed an architecture for the problem of learning generalized task knowledge from single assembly demonstrations. While there are a number of components of the architecture that need to be fully implemented (particularly the perceptual system), we believe that the challenges are just on the fringe of existing technology. Our small user study suggests there is significant utility in having automated assembly instructors. Developing a learning system to acquire the necessary knowledge is perhaps the only practical way to develop a large library of assembly tasks for such systems.

From a machine-learning perspective, our system architecture suggests a view of machine learning that is far from the norm. In particular, learning is a process of interacting with rich knowledge sources and data, in order to arrive at a reliable hypothesis. This suggests a move away from data hungry methods to methods that actively consult knowledge sources as a proxy for hard-to-gather training data. The system described in this report represents only a hardwired instantiation of what we envision such a learning system should look like. In particular, the interaction between the learner and knowledge sources was hardwired and inflexible. Rather, the ultimate goal should be to develop “representations” of knowledge sources and data, which a learner can then reason about in order to decide on how to best interact with the sources in order to achieve the learning objective. Ultimately, we believe that such approaches are necessary if we are to achieve truly integrated intelligent systems with learning capabilities.

Appendix A: Fault-Table Diagnosis for DNS

In fault-table diagnosis, we must define a set of symptoms and a set of faults. The fault table then lists the symptoms of each fault. Diagnostic algorithms can use the fault table to drive efficient diagnosis, particularly if the probability of each fault is known and if the cost of measuring each symptom is known.

The following table lists the symptoms and faults that we were able to identify for a single DNS installation.

Symptom ID	Symptom Description	Fault ID	Fault Description
s1	Loss of Network Connectivity	f1	Forgot to Increment Serial Number
s2	Response from Unexpected Source	f2	Forgot to Reload Primary Master Server after changes are made
s3	Recursion Bugs	f3	Corrupt db.cache data
s4	Client unsure on handling of NS record in Authority Section	f4	Ignored Referral
s5	No answer to query	f5	To Many Referrals
s6	Client calls on server to many times	f6	Malicious Server
s7	Nave Server is infected with bogus cache data	f7	Zero Answer
s8	A server refers to itself in the authority section	f8	Added Name to Database File but Forgot to Add PTR Record
s9	Cache Leaks	f9	Incorrect db.cache file
s10	Remote Names Can't Be Looked Up	f10	db.cache size set to small
s11	Name Error Bugs	f11	Server does not do negative caching
s12	Lookups Take a Long Time	f12	Syntax error in zone data file on master
s13	Wrong or Inconsistent Answer	f13	Incorrect IP address for master on Slave zone data file
s14	Slave name server data does note change when change is made	f14	Syntax error in configuration file or zone data file
s15	Name Server Keeps Loading Old Data	f15	Missing Dot at the End of a Domain Name in a Zone Data file
s16	Is invalid proceeding anyway	f16	Missing root.hints or db.root data
s17	Slave Server Can't Load Zone Data	f17	Missing Subdomain Delegation
s18	Internet services refused	f18	TTL exceeded
s19	Host fails authentication checks	f19	Syntax error in resolv.conf
s20	Inconsistent or Missing Bad Data	f20	Incorrect labels in DNS name
s21	Lame-server reported	f21	Incorrect SOA format
s22	Name Server fails to load	f22	Incorrect Glue records
s23	Name server reports Too Many Open Files	f23	Retry Interval is to low
		f24	Incorrect address in query list allow-query {address_match_list }
		f25	Incorrect configuration named.conf listen-on {ip_address }
		f26	PTR record points to CNAME
		f27	Expire time exceeded
		f28	Loss of Network Connectivity

The following table is the fault-symptom table. A “1” in a cell indicates that the fault

Faults

Symptoms	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20	f21	f22	f23	f24	f25	f26	f27	f28
s1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s2	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
s3	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s5	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	1	1	0	0	1	0
s6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
s7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
s8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s9	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s10	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	0	0	1	1
s11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
s12	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0
s13	0	1	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0
s14	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s15	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s17	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
s18	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
s19	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
s20	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0
s21	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
s22	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Appendix B: Machine Learning for Cognitive Networks: Technology Assessment and Research Challenges

Tom Dietterich
Department of Computer Science
Oregon State University
Corvallis, OR 97331

Pat Langley
Institute for the Study of Learning and Expertise
2164 Staunton Court
Palo Alto, CA 94306

Abstract

The field of machine learning has made major strides over the last 20 years. This document summarizes the major problem formulations that the discipline has studied, then reviews three tasks in cognitive networking and briefly discusses how aspects of those tasks fit these formulations. After this, it discusses challenges for machine learning research raised by Knowledge Plane applications and closes with proposals for the evaluation of learning systems developed for these problems.

B1. Background and Motivation

Recently, Clark (2002) and Partridge (2003) have proposed a new vision for computer network management—the Knowledge Plane—that would augment the current paradigm of low-level data collection and decision making with higher-level processes. One key idea is that the Knowledge Plane would learn about its own behavior over time, making it better able to analyze problems, tune its operation, and generally increase its reliability and robustness. This suggests the incorporation of concepts and methods from machine learning (Langley, 1995; Mitchell, 1997), an established field that is concerned with such issues.

Machine learning aims to understand computational mechanisms by which experience can lead to improved performance. In everyday language, we say that a person has `learned' something from an experience when he can do something he could not, or could not do as well, before that experience. The field of machine learning attempts to characterize how such changes can occur by designing, implementing, running, and analyzing algorithms that can be run on computers. The discipline draws on ideas from many other fields, including statistics, cognitive psychology, information theory, logic, complexity theory, and operations research, but always with the goal of understanding the computational character of learning.

There is general agreement that representational issues are central to learning. In fact, the field is often divided into paradigms that are organized around representational formalisms, such as decision trees, logical rules, neural networks, case libraries, and probabilistic notations. Early debate revolved around which formalism provided the best support for machine learning, but the advent of experimental comparisons around 1990 showed that, in general, no formalism led to better learning than any other. However, it also revealed that the specific features or representational encodings mattered greatly, and careful feature engineering remains a hallmark of successful applications of machine learning technology (Langley & Simon, 1995).

Another common view is that learning always occurs in the context of some performance task, and that a learning method should always be coupled with a performance element that uses the knowledge acquired or revised during learning. Figure~14 depicts such a combined system, which experiences the environment, uses learning to transform those experiences into knowledge, and makes that knowledge available to a performance module that operates in the environment. Performance refers to the behavior of the system when learning is disabled. This may involve a simple activity, such as assigning a label or selecting an action, but it may also involve complex reasoning, planning, or interpretation. The general goal of learning is to improve performance on whatever task the combined system is designed to carry out.

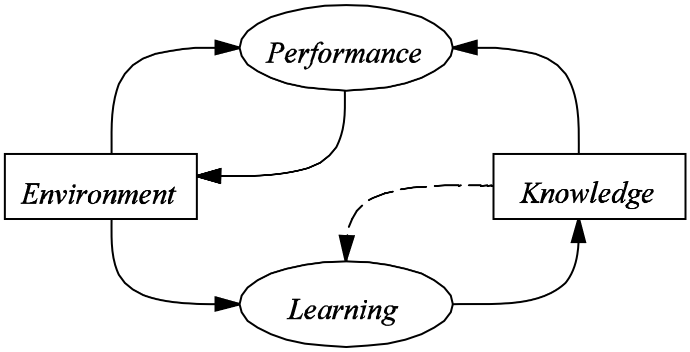


Figure 14 - . Relationship between learning, performance, knowledge, and the environment.

We should clarify a few more points about the relations between learning, performance, and knowledge. The figure suggests that the system operates in a continuing loop, with performance generating experiences that produce learning, which in turn leads to changes in performance, and so on. This paradigm is known as *on-line* learning, and characterizes some but not all research in the area. A more common approach, known as off-line learning, instead assumes that the training experiences are all available at the outset, and that learning transforms these into knowledge only once. The figure also includes an optional link that lets the system's current knowledge influence the learning process. This idea is not widely used in current research, but it can assist learning significantly when relevant knowledge is available.

In this paper, we examine various aspects of machine learning that touch on cognitive approaches to networking. We begin by reviewing the major problem formulations that have been studied in machine learning. Then we consider three tasks that the Knowledge Plane is designed to support and the roles that learning could play in them. Next we discuss some open issues and research challenges that the Knowledge Plane poses for the field of machine learning. Finally, we propose some methods and criteria for evaluating the contribution of machine learning to cognitive networking tasks.

B2. Problem Formulations in Machine Learning

Treatments of machine learning (e.g., Langley, 1995; Mitchell, 1997) typically organize the field along representational lines, depending on whether one encodes learned knowledge using decision trees, neural networks, case libraries, probabilistic summaries, or some other notation. However, a more basic issue concerns how one *formulates* the learning task in terms of the inputs that drive learning and the manner in which the learned knowledge is utilized. This section examines three broad formulations of machine learning.

B2.1 Learning for Classification and Regression

The most common formulation focuses on learning knowledge for the performance task of *classification* or *regression*. Classification involves assigning a test case to one of a finite set of classes, whereas regression instead predicts the case's value on some continuous variable or attribute. In the context of network diagnosis, one classification problem is deciding whether a connection failure is due to the target site being down, the target site being overloaded, or the ISP service being down. An analogous regression problem might involve predicting the time it will take for the connection to return. Cases are typically described as a set of values for discrete or continuous attributes or variables. For example, a description of the network's state might include attributes for packet loss, transfer time, and connectivity. Some work on classification and regression instead operates over relational descriptors. Thus, one might describe a particular situation in terms of node connections and whether numeric attributes at one node are higher than those at an adjacent node.

In some situations, there is no special attribute that one knows at the outset will be predicted from others. Instead, one may need to predict the value of any unobserved attributes in terms of others that have been observed. This performance task, often called pattern completion or flexible prediction, can be used for symbolic attributes, continuous attributes, or a mixture of them. For example, given information about some network variables that are measured easily and cheaply, one might want to predict the values of other network variables that are more expensive to measure. A related task involves predicting the conditional probabilities that different values will occur for unknown variables given observed values for others. Alternatively, one may want to predict the joint probability distribution over the entire space of possible instances.

One can formulate a number of distinct learning tasks that produce knowledge for use in classification or regression. The most common, known as supervised learning, assumes the learner is given training cases with associated classes or values for the attribute to be predicted. For example, one might provide a supervised learning method with 200 instances of four different types of connection failure, say 50 instances of each class, with each instance described in terms of the attributes to be used later during classification. The analogous version for regression would provide instead the time taken to restore the connection for each training instance.

There exist a variety of well-established paradigms for supervised learning, including decision-tree and rule induction (Quinlan, 1993; Clark & Niblett, 1988), neural network methods (Rumelhart et al., 1986), nearest neighbor approaches (Aha et al., 1991), and probabilistic methods (Buntine, 1996). These frameworks differ in the formalisms they employ for representing learned knowledge, as well as their specific algorithms for using and learning that knowledge. What these methods hold in common is their reliance on a target class or response variable to direct their search through the space of predictive models. They also share a common approach to evaluation, since their goal is to induce predictive models from training cases that have low error on novel test cases.

A second broad class of tasks, unsupervised learning, assumes that the learner is given training cases without any associated class information or any specific attribute singled out for prediction. For example, one might provide an unsupervised method with the same 200 instances as before, but not include any information about the type of connection failure or the time taken to restore the connection.

As with supervised learning, there exist many techniques for learning from unsupervised data, but these fall into two broad classes. One approach, known as clustering (Fisher, 1987; Cheeseman et al., 1988), assumes the goal of learning is to assign the training instances to distinct classes of its own invention, which can be used to classify novel instances and make inferences about them, say through pattern completion. For example, a clustering algorithm might group the 200 training instances into a number of classes that represent what it thinks are different types of service interruption. Another approach, known as density estimation (Priebe & Marchette, 1993), instead aims to build a model that predicts the probability of occurrence for specific instances. For example, given the same data about service interruptions, such a method would generate a probability density function that covers both the training instances and novel ones.

A third formulation, known as semi-supervised learning (Blum & Mitchell, 1998), falls between the two approaches we have already discussed. In this framework, some of the training instances come with associated classes or values for predicted attributes, but others (typically the majority) do not have this information. This approach is common in domains such as text classification, where training cases are plentiful but class labels are costly. The goal is similar to that for supervised learning, that is, to induce a classifier or regressor that makes accurate predictions, but also to utilize the unlabeled instances to improve this behavior. For example, even if only 20 of the 200 training instances on service interruption included class information, one might still use regularities in the remaining instances to induce more accurate classifiers.

Classification and regression are the most basic capabilities for which learning can occur. As a result, the field has developed robust methods for these tasks and they have been applied widely to develop accurate and useful predictive models from data. Langley and Simon (1995) review some early successes of these methods, and they have since formed the backbone for many commercial applications within the data-mining movement. Methods for classification and regression learning can also play a role in more complex tasks, but such tasks also introduce other factors that require additional mechanisms, as discussed in the next section.

B2.2 Learning for Acting and Planning

A second formulation addresses learning of knowledge for selecting actions or plans for an agent to carry out in the world. In its simplest form, action selection can occur in a purely reactive way, ignoring any information about past actions. This version has a straightforward mapping onto classification, with alternative actions corresponding to distinct classes from which the agent can choose based on descriptions of the world state. One can also map it onto regression, with one predicting the overall value or utility of each action in a given world state.

Both approaches can also be utilized for problem solving, planning, and scheduling. These involve making cognitive choices about future actions, rather than about immediate actions in the environment. These activities typically involve search through a space of alternatives, which knowledge can be used to constrain or direct. This knowledge may take the form of classifiers for which action to select or regression functions over actions or states. However, it can also be cast as larger-scale structures called macro-operators that specify multiple actions that should be carried out together.

As with classification and regression, one can formulate a number of learning tasks that produce knowledge for action selection and search. The simplest approach, known as a learning apprentice (Mitchell et al., 1985) or an adaptive interface (Langley, 1999), embeds the learner within a larger system that interacts with a human user. This system may accept directions from the user about what choices to make or it may make recommendations to the user, who can then accept them or propose other responses. Thus, the user gives direct feedback to the system about each choice, effectively transforming the problem of learning to select actions into a supervised learning task, which can then be handled using any of the methods discussed earlier. A related paradigm, known as programming by demonstration (Cypher, 1993), focuses on learning macro-operators for later invocation by the user to let him accomplish things in fewer steps.

For example, one might implement an interactive tool for network configuration that proposes, one step at a time, a few alternative components to incorporate or connections among them. The human user could select from among these recommendations or reject them all and select another option. Each such interaction would generate a training instance for use in learning how to configure a network, which would then be used on future interactions. One can imagine similar adaptive interfaces for network diagnosis and repair.

A closely related formulation of action learning, known as behavioral cloning (Sammut et al., 1992), collects traces of a human acting in some domain, but does not offer advice or interact directly. Again, each choice the human makes is transformed into a training case for use by supervised learning. The main difference is that behavioral cloning aims to create autonomous agents for carrying out a sequential decision-making task, whereas learning apprentices and adaptive interfaces aim to produce intelligent assistants. For example, one could watch a human expert execute a sequence of commands in configuring a computer network, transform these into supervised training cases for learning which actions to select or estimating the value of available choices. However, one might also attempt to extract, from the same trace, recurring sets of actions for composition into macro-operators that would let one solve the same problem in fewer steps.

A somewhat different formulation involves the notion of learning from delayed reward, more commonly known as reinforcement learning. Here the agent typically carries out action to take in the environment and receives some reward signal that indicates the desirability of the resulting states. However, because many steps may be necessary before the agent reaches a desirable state (e.g., reestablishing a service connection), the reward can be delayed. Research in the reinforcement learning framework falls into two main paradigms. One casts control policies in terms of functions that map state descriptions and available actions onto expected values (Kaelbling et al., 1996; Sutton & Barto, 1998). This approach involves propagating rewards backward over action sequences to assign credit, and may invoke a regression method to learn a predictor for expected values. Another paradigm instead encodes control policies as a more direct mapping from state descriptions onto actions, with learning involving search through the space of such policies (Williams, 1992, Moriarty et al., 1999).

For instance, one might apply either approach to learning policies for dynamic network routing (Boyan & Littman, 1994). The reward signal here might be based on the standard metrics for route performance. The system would try establishing different routes, each of which involves a number of decision-making steps, and learn routing policies based on the observed performance. Over time, the routes selected by the learned policy would change, giving improved behavior for the overall network.

Another formulation is closely related to reinforcement learning, but involves *learning from problem solving* and mental search (Sleeman et al., 1982), rather than from actions in the environment. Here the agent has some model of the effects of actions or the resources they require which it can use to carry out mental simulations of action sequences. However, there typically exist many possible sequences, which introduces the problem of search through a problem space. Such search can produce one or more sequences that solve the problem, but it also generate dead ends, loops, and other undesirable outcomes. Both successes and failures provide material for learning, in that they distinguish between desirable and undesirable choices, or at least suggest relative desirability.

Research on learning from problem-solving traces occurs within a three broad paradigms. Some work focuses on learning local search-control knowledge for selecting, rejecting, or preferring actions or states. This knowledge may be cast as control rules or some related symbolic representation, or it may be stated as a numeric evaluation function. The latter approach is closely related to methods for estimating value functions from delayed reward, which has occasionally been used for tasks like scheduling (Zhang & Dietterich, 1995) and integrated circuit layout (Boyan & Moore, 2000). Another paradigm emphasizes the formation from solution paths of macro-operators that take larger steps through the problem space in order to reduce the effective depth of search. A third framework, analogical problem solving, also stores large-scale structures, but utilizes them in a more flexible manner by adapting them to new problems.

For example, one might apply any of these approaches to tasks like network routing and configuration. Such an application would require some model of the effects that individual choices would produce, so that the agent can decide whether a given state is desirable before actually generating it in the world. Thus, the system would start with the ability to generate routes or configurations, but it might do this very inefficiently if the search space is large. After repeated attempts at routing or configuration, it would acquire heuristic knowledge about how to direct its search, letting it produce future solutions much more efficiently without loss in quality.

A final formulation involves the empirical optimization of a complex system. Consider the problem of adjusting a chemical plant's parameters to improve its performance (e.g., reduce energy consumption, reduce waste products, increase product quality, increase rate of production, and so forth). If a predictive model of the plant is not available, the only recourse may be to try various settings of the parameters and see how the plant responds.

One example of this idea, response surface methodology (Myers & Montgomery, 1995) attempts to find the optimal operating point of a system by measuring system behavior at various points. The classic method executes a classical experiment design (e.g., some form of factorial design) about the current operating point and fits the results with a quadratic function to estimate the local shape of the objective function surface. Then it chooses a new operating point at the optimum of that quadratic surface and repeats the process.

Machine learning researchers have studied methods that make weaker assumptions and require fewer training examples. One approach (Moore et al., 1998) employs regression to analyze the results of previous experiments and determine a region of interest in which the objective function can be approximated well, then chooses a new test point that is distant from other test points while still lying within this region. An alternative approach (Baluja & Caruana, 1995) is more appropriate for searching discrete parameter spaces such as those that arise in network configuration. Given a set of parameter settings (configurations) for which the performance has been measured, one fits a probability distribution to predict where additional ``good'' points are located, then samples a new set of configurations according to that distribution, measures their performance, and continues until convergence.

Before closing, it is worth making two other points about learning for action selection and planning. First, in many domains, sensing requires active invocation, so that one can view it as a kind of action. Thus, an agent can learn policies for sensing, say to support efficient network diagnosis, just as it can for effectors, such as closing down a link in response to a suspected attack. Second, some methods for plan learning assume the availability of action models that describe the expected effects when actions are invoked. This leads in turn to the task of learning such action models from observations. This has many similarities to the problem of classification and regression learning, but aims to support higher-level learning about policies for acting and planning.

B2.3 Learning for Interpretation and Understanding

A third formulation focuses on learning knowledge that lets one interpret and understand situations or events. Classification can be seen as a simple example of this idea, since one can `understand' an instance as being an example of some class. However, more sophisticated approaches attempt to interpret observations in a more constructive manner, by combining a number of separate knowledge elements to explain them. The key difference is that classification and regression are content with models that make accurate predictions, whereas interpretive approaches require models that explain the data in terms of deeper structures. This process of explanation generation is often referred to as *abduction*.

The explanatory or abductive approach is perhaps most easily demonstrated in natural language processing, where a common performance task involves parsing sentences using a context-free grammar or some related formalism. Such a grammar contains rewrite rules that refer to nonterminal symbols for types of phrases and parts of speech, and a parse tree specifies how one can derive or explain a sentence in terms of these rules. One can apply similar ideas to other domains, including the interpretation and diagnosis of network behavior. For example, given anomalous data about the transfer rates between various nodes in a network, one might explain these observations using known processes, such as demand for a new movie that is available at one site and desired by others.

One can state a number of different learning tasks within the explanatory framework. The most tractable problem assumes that each training case comes with an associated explanation cast in terms of domain knowledge. This formulation is used commonly within the natural language community, where the advent of `tree banks' has made available large corpora of sentences with their associated parse trees. The learning task involves generalizing over the training instances to produce a model that can be used to interpret or explain future test cases. Naturally, this approach places a burden on the developer, since it requires hand construction of explanations for each training case, but it greatly constrains the learning process, as it effectively decomposes the task into a set of separate classification or density estimation tasks, one for each component of the domain knowledge.

A second class of learning task assumes that training instances do not have associated explanations, but provides background knowledge from which the learner can construct them. This problem provides less supervision than the first, since the learner must consider alternative explanations for each training case and decide which ones are appropriate. However, the result is again some model that can be applied to interpret or explain future instances. This formulation is less burdensome on the developer, since he need not provide explanations for each training case, but only a domain theory from which the learner can construct them itself. Flann and Dietterich (1989) have referred to this learning task as induction over explanations, but it is also closely related to some work on constructive induction (Drastal et al., 1989).

A final variant on learning for understanding provides training cases with neither explanations nor background knowledge from which to construct them. Rather, the learner must induce its own explanatory structures from regularities in the data, which it can then utilize to interpret and understand new test instances. An example from natural language involves the induction of context-free grammars, including both nonterminal symbols and the rewrite rules in which they occur, from legal training sentences (Stolcke & Omohundro, 1994). Clearly, this task requires even less effort on the developer's part, but places a greater challenge on the learning system. This approach has gone by a variety of names in the machine learning literature, including term generation, representation change, and constructive induction (though this phrase has also been used for the second task).

Because learning tasks that produce explanatory models are generally more difficult than those for classification and regression, some researchers have formulated more tractable versions of them. One variant assumes the qualitative structure of the explanatory model is given and that learning involves estimating numeric parameters from the data. Examples of this approach include determining the probabilities in a stochastic context-free grammar, tuning the parameters in sets of differential equations, and inferring conditional probabilities in a Bayesian network. Another variation, known as theory revision, assumes an initial explanatory model that is approximately correct and utilizes training data to alter its qualitative structure. Examples include revising Horn clause programs from classified training cases, improving sets of equations from quantitative data, and altering grammars in response to training sentences.

B2.4 Summary of Problem Formulations

In summary, one can formulate machine learning tasks in a variety of ways. These differ in both the manner in which learned knowledge is utilized and, at a finer level, in the nature of the training data that drives the learning process. Table B1 summarizes the main formulations that have been discussed in this section. However, it is important to realize that different paradigms have received different degrees of attention within the machine learning community. Supervised approaches to classification and regression have been the most widely studied by far, with reinforcement learning being the second most common. Yet their popularity in the mainstream community does not imply they are the best ways to approach problems in computer networking, and research on the Knowledge Plane should consider all the available options.

Table B1: Summary of Machine Learning Problem Formulations

Formulation	Performance Task
Classification and regression	Predict *y* given *x* Predict rest of *x* given part of *x* Predict *P(x)* given *x*
Acting and planning	Iteratively choose action *a* in state *s* Choose actions ‹a_1,…,a_n› to achieve goal *g* Find setting *s* to maximize objective *J(s)*
Interpretation and understanding	Parse data stream into tree structure of objects or events

Another important point is that one can often formulate a given real-world problem as a number of quite different learning tasks. For example, one might cast diagnosis of network faults as a classification problem that involves assigning the current network state to either a normal condition or one of a few prespecified faulty conditions. However, one could instead formulate it as a problem of understanding anomalous network behavior, say in terms of unobservable processes that, taken together, can explain recent statistics. Yet another option would be to state diagnosis as a problem of selecting active sensors that narrow down alternatives. Each formulation suggests different approaches to the diagnostic task, to learning knowledge in support of that task, and to criteria for evaluating the success of the learning component.

B3. Tasks in Cognitive Networking

The vision for the Knowledge Plane (Clark, 2002; Partridge, 2003) describes a number of novel capabilities for computer networks. This section reviews three capabilities that the vision assumes in terms of the cognitive functionalities that are required. These include anomaly detection and fault diagnosis, responding to intruders and worms, and rapid configuration of networks.

B3.1 Anomaly Detection and Fault Diagnosis

Current computer networks require human managers to oversee their behavior and ensure that they deliver the services desired. To this end, the network managers must detect unusual or undesirable behaviors, isolate their sources, diagnose the fault, and repair the problem. These tasks are made more challenging because large-scale networks are managed in a distributed manner, with individuals having access to information about, and control over, only portions of the system. Nevertheless, it will be useful to examine the activities in which a single network manager engages.

The first activity, anomaly detection, involves the realization that something unusual or undesirable is transpiring within the network. One possible approach to this problem, which applies recent advances in Bayesian networks, is to formulate it as a density estimation problem. Individual components, larger regions of the network, or, at some level, the entire internet could be modeled as the joint probability distribution of various quantities (queue lengths, traffic types, round-trip-times, and so on). An anomaly is defined as a low probability state of the network.

Another possible approach is sometimes called one-class learning or learning a characteristic description of a class. A classifier can be learned that attempts to find a compact description that covers a target percentile (e.g., 95%) of the ``normal'' traffic. Anything classified as ``negative'' by this classifier can then be regarded as an anomaly.

There are several issues that arise in anomaly detection. First, we must choose the level of analysis and the variables to monitor for anomalies. This may involve first applying methods for interpreting and summarizing sensor data. In the Knowledge Plane, one can imagine having whole hierarchies of anomaly detectors looking for changes in the type of network traffic (e.g., by protocol type), in routing, in traffic delays, in packet losses, in transmission errors, and so on. Anomalies may be undetectable at one level of abstraction but easy to detect at a different level. For example, a worm might escape detection at the level of a single host, but be detectable when observations from several hosts are combined.

The second issue is the problem of false alarms and repeated alarms. Certain kinds of anomalies may be unimportant, so network managers need ways of training the system to filter them out. Supervised learning methods could be applied to this problem.

The second activity, fault isolation, requires the manager to identify the locus of an anomaly or fault within the network. For example, if a certain route has an especially heavy load, this may be due to changes at a single site along that route rather than to others. Hence, whereas anomaly detection can be performed locally (e.g., at each router), fault isolation requires the more global capabilities of the Knowledge Plane to determine the scope and extent of the anomaly.

The activity of diagnosis involves drawing some conclusions about the cause of the anomalous behavior. Typically, this follows fault isolation, although in principle one might infer the presence of a specific problem without knowing its precise location. Diagnosis may involve the recognition of some known problems; say one the network manager has encountered before, or the characterization of a new problem that may involve familiar components.

Supervised learning methods can be applied to allow a network manager to teach the system how to recognize known problems. This could be a prelude to automatically solving them, as discussed below.

Both fault isolation and diagnosis may require active measurements to gather information. For example, an anomaly found at a high level of aggregation would typically require making more detailed observations at finer levels of detail to understand the cause. In the ``Why?'' scenario, one can imagine active probes of both the local computer (e.g., its configuration) and the internet (e.g., ``pings'' to see if the destination is reachable and up). Diagnosis usually must balance the cost of gathering information against the potential informativeness of the action. For example, if the ping succeeds, it requires little time, but otherwise it can take much longer to time out. If our goal is to diagnose the problem as quickly as possible, then ping might be a costly action to perform. (Recent work in an area known as ``cost-sensitive learning'' addresses this tradeoff between cost and informativeness.)

Fault isolation and diagnosis also typically require models of the structure of the system under diagnosis. Much recent effort in network research has sought to provide better ways of understanding and visualizing the structure of the internet. Machine learning for interpretation could be applied to help automate this process. The resulting structural and behavioral models could then be used by model-based reasoning methods to perform fault isolation and diagnosis.

Once a network manager has diagnosed a problem, he is in a position to repair it. However, there may exist different courses of action that would eliminate the problem, which have different costs and benefits. Moreover, when multiple managers are involved in the decision, different criteria may come into play that leads to negotiation. Selecting a repair strategy requires knowledge of available actions, their effects on network behavior, and the tradeoffs they involve.

Supervised learning methods could be applied to learn the effects of various repair actions. Methods for learning in planning could be applied to learn repair strategies (or perhaps only to evaluate repair strategies suggested by a human manager). There may be some opportunity here for ``collaborative filtering'' methods that would provide an easy way for managers to share repair strategies.

As stated, the `Why' problem (Clark, 2002; Partridge, 2003) requires diagnosis of an isolated fault, but one can imagine variations that involve answering questions about anomalies, fault locations, and actions taken to repair the system. Each of these also assumes some interface that lets the user pose a specific question in natural language or, more likely, in a constrained query language. Defining the space of Why questions the Knowledge Plane should support is an important research task that deserves attention early in the research program.

B3.2 Responding to Intruders and Worms

Responding to intruders (human, artificial, or their combination) and keeping networks and applications safe encompasses a collection of tasks that are best explained depending on the time at which they are performed by a network manager. We can group them into tasks that occur before, during or after the occurrence of an intrusion, as the temporal model in Figure 15 depicts.

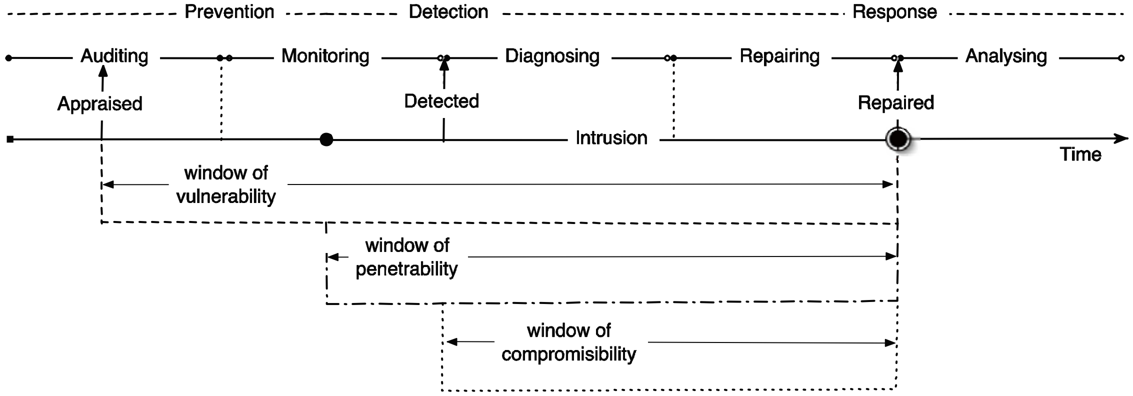


Figure 15 - : Time axis model of incident prevention, detection, and response tasks.

Prevention Tasks. Network managers try to minimize the likeliness of future intrusions by constantly auditing the system and eliminating threats beforehand. A network manager proactively performs security audits testing the computer systems for weaknesses—vulnerabilities or exposures. However scan tools (i.e., Nessus, Satan, and Oval) used for penetration or vulnerability testing only recognize a limited number of vulnerabilities given the ever increasing frequency of newly detected possibilities for breaking into a computer system or disturbing its normal operation. Thus, network managers continuously update scan tools with new plug-ins that permit them to measure new vulnerabilities. Once the existence of a vulnerability or exposure is perceived, network managers assess the convenience of discontinuing the service or application affected until the corresponding patch or intrusion detection signature is available. A tradeoff between risk level and service level is made in every assessment.

Network managers aim at shrinking the window of vulnerability, the time gap between when a new vulnerability or exposure is discovered and a preventing solution (patch, new configuration, etc.) is provided, as much as possible. A basic strategy to accomplish that objective is based on two conservative tasks: first, minimizing the number of exposures (i.e., disable unnecessary or optional services by configuring firewalls to allow only the use of ports that are necessary for the site to function) and, second, increasing awareness of new vulnerabilities and exposures (i.e., the subscription model that Partridge discusses with relation to worms).

Finally, network managers continuously monitor the system so that pre-intrusion behavioral patterns can be understood and used for further reference when an intrusion occurs. Monitoring is an ongoing, preventive task.

Detection Tasks. The sooner an intrusion is detected, the more chances there are for impeding an unauthorized use or misuse of the computer system. Network managers monitor computer activities at different levels of detail: system call traces, operating system logs, audit trail records, resource usage, network connections, etc. Normally, they constantly try to fuse and correlate real-time reports and alerts stemming from different security devices (e.g., firewalls and intrusion detection systems) to stop suspicious activities before they have a negative impact (i.e., degrading or disrupting operations).

Different sources of evidence are valuable given the evolving capabilities of intruders to elude security devices. The degree of suspicion and malignancy associated to each report or alert still requires continuous human oversight. Consequently, network managers are continually overwhelmed with a vast amount of log information and bombarded with countless alerts. To deal with this onslaught, network managers often tune security devices to provide an admissible number of false alerts even though this increases the risk of not detecting real intrusions.

The time at which an intrusion is detected directly affects the level of damage that an intrusion causes. An objective of network managers is to reduce the *window of penetrability*, the time span that initiates when a computer system has been broken into and extends until the damage has been completely repaired. The correct diagnosis of an intrusion allows a network manager to initiate the most convenient response. However, a tradeoff between quality and rapidness is made in every diagnostic.

Response and Recovery Tasks. As soon as a diagnostic on an intrusion is available, network managers initiate a considered response. This response tries to minimize the impact on the operations (i.e., do not close all ports in a firewall if only blocking one IP address is enough). Network managers try to narrow the *window of compromisibility* of each intrusion—the time gap that starts when an intrusion has been detected and ends when the proper response has taken effect—deploying automatic intrusion response systems. Nevertheless, these systems are still at an early stage and even fail at providing assistance in manual responses. Therefore, network managers employ a collection of ad-hoc operating procedures that indicate how to respond and recover from a type of intrusion. The responses to an attack range from terminating a user job or suspending a session to blocking an IP address or disconnecting from the network to disable the compromised service or host. Damage recovery or *repairing* often entails maintaining the level of service while the system is being repaired, which makes this process difficult to automate. Once the system in completely recovered from an intrusion, network managers collect all possible data to thoroughly analyze the intrusion, trace back what happened, and evaluate the damage. Thus, system logs are continuously backed up. The goal of *post-mortem analysis* is twofold. On the one hand, it gathers forensic evidence (contemplating different legal requirements) that will support legal investigations and prosecution and, on the other hand, it compiles experience and provides documentation and procedures that will facilitate the recognition and repelling of similar intrusions in the future.

Ideally, the ultimate goal of a network manager is to make the three windows (vulnerability, penetrability, and compromisibility) of each possible intrusion converge into a single point in time. Tasks for responding to intruders (human, artificial or a combination of both) should not differ significantly from those tasks needed to recover from non-malicious errors or failures.

B3.3 Network Configuration and Optimization

Network configuration and optimization can be viewed as an instance of the general problem of designing and configuring a system. In this section, we review the space of configuration problems and briefly describe the methods that have been developed in AI and machine learning to solve these problems.

B3.3.1 A Spectrum of Configuration Tasks

The problem of the design and configuration of engineered systems has been studied in artificial intelligence since the earliest days (Tonge, 1963). Configuration is generally defined as a form of routine design from a given set of components or types of components (i.e., as opposed to designing the components themselves). As such, there is a spectrum of configuration problems of increasing difficulty, as shown in Table B2. The simplest task is *parameter selection*, where values or chosen for a set of global parameters in order to optimize some global objective function. Two classic examples are the task of setting the temperature, cycle time, pressure, and input/output flows of a chemical reactor and the task of controlling the rate of cars entering a freeway and the direction of flow of the express lanes. If a model of the system is known, this becomes purely an optimization problem, and many algorithms have been developed in operations research, numerical analysis, and computer science to solve such problems.

The second task is compatible parameter selection. Here, the system consists of a set of components that interact with one another to achieve overall system function according to a fixed topology of connections. The effectiveness of the interactions is influenced by parameter settings which must be compatible in order for sets of components to interact. For example, a set of hosts on a subnet must agree on the network addresses and subnet mask in order to communicate using IP. Global system performance can depend in complex ways on local configuration parameters. Of course, there may also be global parameters to select as well, such as the protocol family to use.

The third task is topological configuration. Here, the system consists of a set of components, but the topology must be determined. For example, given a set of hosts, gateways, file servers, printers, and backup devices, how should the network be configured to optimize overall performance? Of course, each proposed topology must be optimized through compatible parameter selection.

Finally, the most general task is component selection and configuration. Initially, the configuration engine is given a catalog of available types of components (typically along with prices), and it must choose the types and quantities of components to create the network (and then, of course, solve the Topological Configuration problem of arranging these components).

Table B2: Configuration tasks in increasing order of complexity

Problem:	Global parameters	Local parameters	Topology	Components
Global parameter configuration	XX			
Compatible parameter configuration	XX	XX		
Topological configuration	XX	XX	XX	
Component selection and configuration	XX	XX	XX	XX

B3.3.2 The Reconfiguration Process

The discussion thus far has dealt only with the problem of choosing a configuration. However, a second aspect of configuration is determining how to implement the configuration efficiently. When a new computer network is being installed (e.g., at a trade show), the usual approach is to install the gateways and routers; then the file and print servers; and finally individual hosts, network access points, and the like. The reason for this is that this order makes it easy to test and configure each component and it minimizes the amount of re-work. Automatic configuration tools (e.g., DHCP) can configure the individual hosts if the servers are in place first.

A different challenge arises when attempting to change the configuration of an existing network, especially if the goal is to move to the new configuration without significant service interruptions. Most configuration steps require first determining the current network configuration, and then planning a sequence of reconfiguration actions and tests to move the system to its new configuration. Some steps may cause network partitions that prevent further (remote) configuration. Some steps must be performed without knowing the current configuration (e.g., because there is already a network partition, congestion problem, or attack).

B3.3.3 Existing AI/ML Work on Configuration

Parameter Selection. As we discussed above, parameter selection becomes optimization (possibly difficult, non-linear optimization) if the model of the system is known. Statisticians have studied the problem of empirical optimization in which no system model is available.

Compatible Parameter Configuration. The standard AI model of compatible parameter configuration is known as the *constraint satisfaction problem* (CSP). This consists of a graph where each vertex is a variable that can take values from set of possible values and each edge encodes a pair-wise constraint between the values of the variables that it joins. A large family of algorithms has been developed for finding solutions to CSPs efficiently (Kumar, 1992). In addition, it is possible to convert CSPs into Boolean satisfiability problems, and very successful randomized search algorithms, such as WalkSAT (Selman et al., 1993), have been developed to solve these problems.

The standard CSP has a fixed graph structure, but this can be extended to include a space of possible graphs and to permit continuous (e.g., linear algebraic) constraints. The field of constraint logic programming (CLP; Jaffar & Maher, 1994) has developed programming languages based on ideas from logic programming that have a constraint solver integrated as part of the run-time system. The logic program execution can be viewed as conditionally expanding the constraint graph, which is then solved by the constraint system. Constraint logic programming systems have been used to specify and solve many kinds of configuration problems.

To our knowledge, there has been no work on applying machine learning to help solve compatible parameter configuration problems. There is a simple form of learning that has been applied to CSPs called “no good learning”, but it is just a form of caching to avoid wasting effort during CSP search. There are many potential learning problems including learning about the constraints relating pairs of variables and learning how to generalize CSP solutions across similar problems.

Topological Configuration. Two principal approaches have been pursued for topological configuration problems: refinement and repair. Refinement methods start with a single ``box" that represents the entire system to be configured. The box has an attached formal specification of its desired behavior. Refinement rules analyze the formal specification and replace the single box with two or more new boxes with specified connections. For example, a small office network might initially be specified as a box that connects a set of workstations, a file server, and two printers to a DSL line. A refinement rule might replace this box with a local network (represented as a single box connected to the various workstations and servers) and a router/NAT box. A second refinement rule might then refine the network into a wireless access point and a set of wireless cards (or alternatively, into an Ethernet switch and a set of Ethernet cards and cables). There has been some work on applying machine learning to learn refinement rules in the domain of VLSI design (Mitchell et al., 1985).

The repair-based approach to topological configuration starts with an initial configuration (which typically does not meet the required specifications) and then makes repairs to transform the configuration until it meets the specifications. For example, an initial configuration might just connect all computers, printers, and other devices to a single Ethernet switch, but this switch might be very large and expensive. A repair rule might replace the switch with a tree of smaller, cheaper switches. Repair-based approaches make sense when the mismatch between specifications and the current configuration can be traced to local constraint violations. A repair rule can be written that ``knows how" to repair each kind of violation. Repair-based methods have been very successful in solving scheduling problems (Zweben et al., 1994).

Machine learning approaches to repair-based configuration seek to learn a heuristic function *h(x)* that estimates the quality of the best solution reachable from configuration *x* by applying repair operators. If *h* has been learned correctly, then a hill climbing search that chooses the repair giving the biggest improvement in *h* will lead us to the global optimum. One method for learning *h* is to apply reinforcement learning techniques. Zhang and Dietterich (1995) learned heuristics for optimizing space shuttle payload scheduling; Boyan and Moore (2000) learned heuristics for configuring the functional blocks on integrated circuit chips.

In both refinement and repair-based methods, constraint satisfaction methods are typically applied to determine good parameter values for the current proposed configuration. If no satisfactory parameter values can be found, then a proposed refinement or repair cannot be applied, and some other refinement or repair operator must be tried. It is possible for the process to reach a dead end, which requires backtracking to some previous point or restarting the search.

Component Selection and Configuration. The refinement and repair-based methods described above can also be extended to handle component selection and configuration. Indeed, our local network configuration example shows how refinement rules can propose components to include in the configuration. Similar effects can be produced by repair operators.

Changing Operating Conditions. The methods discussed so far only deal with the problem of optimizing a configuration under fixed operating conditions. However, in many applications, including networking, the optimal configuration may need to change as a result of changes in the mix of traffic and the set of components in the network. This raises the issue of how data points collected under one operating condition (e.g., one traffic mix) and be used to help optimize performance under a different operating condition. To our knowledge, there is no existing research on this question.

B4. Open Issues and Research Challenges

Most research in the field of machine learning has been motivated by problems in pattern recognition, robotics, medical diagnosis, marketing, and related commercial areas. This accounts for the predominance of supervised classification and reinforcement learning in current research. The networking domain requires several shifts in focus and raises several exciting new research challenges, which we discuss in this section.

B4.1 From Supervised to Autonomous Learning

As we have seen above, the dominant problem formulation in machine learning is supervised learning, where a ``teacher" labels the training data to indicate the desired response. While there are some potential applications of supervised learning in KP applications (e.g., for recognizing known networking misconfigurations and intrusions), there are many more applications for autonomous learning that does not require a teacher. In particular, many of the networking applications require looking for anomalies in real-time data streams, which can be formulated as a combination of unsupervised learning and learning for interpretation.

Anomaly detection has been studied in machine learning, but usually it has considered only a fixed level of abstraction. For networking, there can be anomalies at the level of individual packets, but also at the level of connections, protocols, traffic flows, and network-wide disturbances. A very interesting challenge for machine learning is to develop methods that can perform simultaneous unsupervised learning at all of these levels of abstraction. At very fine levels of detail, network traffic is constantly changing, and therefore, is constantly novel. The purpose of introducing levels of abstraction is to hide unimportant variation while exposing important variation.

Anomaly detection at multiple levels of abstraction can exploit regularities at these multiple levels to ensure that the anomaly is real. A similar idea—multi-scale analysis—has been exploited in computer vision, where it is reasonable to assume that a real pattern will be observable at multiple levels of abstraction. This helps reduce false alarms.

B4.2 From Off-Line to On-Line Learning

Most applications of machine learning involve off-line approaches, where data is collected, manually labeled, and then provided to the learning algorithm in a batch process. KP applications involve the analysis of real-time data streams, and this poses new challenges and opportunities for learning algorithms.

In the batch framework, the central constraint is usually the limited amount of training data. In contrast, in the data stream setting, new data is available at every time instant, so this problem is less critical. (Nonetheless, even in a large data stream, there may be relatively few examples of a particular phenomenon of interest, so the problem of sparse training data is not completely eliminated.)

Moreover, the batch framework assumes that the learning algorithm has essentially unlimited amounts of computing time to search through the space of possible knowledge structures. In the on-line setting, the algorithm can afford only a fixed and limited amount of time to analyze each data point.

Finally, in the batch framework, the criterion to be minimized is the probability of error on new data points. In the on-line framework, it makes more sense to consider the response time of the system. How many data points does it need to observe before it detects the relevant patterns? This can be reformulated as a mistake-bounded criterion: how many mistakes does the system make before it learns to recognize the pattern?

B4.3 From Fixed to Changing Environments

Virtually all machine learning research assumes that the training sample is drawn from a stationary data source—the distribution of data points and the phenomena to be learned are not changing with time. This is not true in the networking case. Indeed, the amount of traffic and the structure of the network are changing continuously. The amount of traffic continues to rise exponentially and new autonomous systems are added to the internet almost every day. New networking applications (including worms and viruses) are introduced frequently.

Research in machine learning needs to formalize new criteria for evaluating of learning systems in order to measure success in these changing environments. A major challenge is to evaluate anomaly detection systems, because by definition they are looking for events that have never been seen before. Hence, they cannot be evaluated on a fixed set of data points, and measures are needed to quantify the degree of novelty of new observations.

B4.4 From Centralized to Distributed Learning

Another important way in which KP applications differ from traditional machine learning problems is that, in the latter, it has usually been possible to collect all of the training data on a single machine and run the learning algorithm over that data collection. In contrast, a central aspect of the Knowledge Plane is that it is a distributed system of sensors, anomaly detectors, diagnostic engines, and self-configuring components.

This raises a whole host of research issues. First, individual anomaly detectors can form models of their local traffic, but they would benefit from important traffic models learned elsewhere in the KP. This would help them detect a new event the first time they see it, rather than having to be exposed multiple times before the event pattern emerges.

Second, some events are inherently distributed patterns of activity that cannot be detected at an individual network node. The research challenge here is to determine what kinds of statistics can be collected at the local level and pooled at the regional or global level to detect these patterns. This may involve a bi-directional process of information exchange in which local components report summary statistics to larger-scale "think points". These think points detect a possible pattern that requires additional data to verify. So they need to request the local components to gather additional statistics. Managing this bi-directional statistical reasoning is an entirely new topic for machine learning research.

B4.5 From Engineered to Constructed Representations

An important ingredient in the success of existing learning systems is the careful engineering of the attributes describing the training data. This "feature engineering" process is not well understood, but it involves combining background knowledge of the application domain with knowledge about learning algorithms. To illustrate this, consider a very simple example in networking arises in intrusion detection: Rather than describing network traffic using absolute IP addresses, it is better to describe packets according to whether they share the same or different IP addresses. This ensures that the learned intrusion detector is not specific to a single IP address but instead looks for patterns among a set of packets sharing a common address, regardless of the absolute value of the address.

A critical challenge for machine learning is to develop more automatic ways of constructing the representations given to the learning algorithms. This requires making explicit the design principles currently used by human data analysts.

B4.6 From Knowledge-Lean to Knowledge-Rich Learning

An important factor influencing the development of machine learning has been the relative cost of gathering training data versus building knowledge bases. The constructing and debugging of knowledge bases is a difficult and time-consuming process, and the resulting knowledge bases are expensive to maintain. In contrast, there are many applications where training data can be gathered fairly cheaply. This is why speech recognition and optical character recognition systems have been constructed primarily from training data. Any normal adult human is an expert in speech recognition and optical character recognition, so it is easy for them to label data points to training a learning system.

There are other domains (including networking), where there are very few experts available, and their time is perhaps better employed in developing formal representations of the knowledge they possess about network architectures and configurations. This is particularly true in the area of network diagnosis and configuration, where experts can help construct models of network components and prescribe rules for correct configuration. This raises the challenge of how to combine training data with human-provided models and rules. This should become an important goal for future machine learning research.

B4.7 From Direct to Declarative Models

Most machine learning systems seek to induce a function that maps directly from inputs to outputs and therefore requires little inference at run time. In an optical character recognition system, for example, the learned recognizer takes a character image as input and produces the character name as output without any run-time inference. We will call this “direct knowledge”, because the learned knowledge performs the task directly. However, as applications become more complex, a simple view of the performance element as a classifier (or direct decision maker) is no longer adequate. Diagnosis and configuration tasks require a more complex performance element that makes a sequence of interacting decisions at run time. These performance elements typically require declarative knowledge such as “what is the probability that misconfigured gateway will exhibit symptom X?” or “it is illegal to simultaneously select configuration options Y and Z.” An important goal for machine learning is to learn these forms of declarative knowledge (i.e., knowledge that makes minimal assumptions about how it will be used by the performance element).

Declarative knowledge is easier for people to understand, and it can be more easily combined with human-provided knowledge as well. Hence, acquiring declarative knowledge is an important challenge for machine learning in the context of the Knowledge Plane.

B5. Challenges in Methodology and Evaluation

Machine learning research has a long history of experimental evaluation, with some examples dating back to the 1960s, well before the field was a recognized entity. However, the modern experimental movement began in the late 1980s, when researchers realized the need for systematic comparisons (e.g., Kibler & Langley, 1988) and the first data repository was launched. Other approaches to evaluation, including formal analysis and comparison to human behavior, are still practiced, but, over the past decade, experimentation has come to dominate the literature on machine learning, and we will focus on that approach in our discussions of cognitive networking.

Experimentation involves the systematic variation of independent factors to understand their impact on dependent variables that describe behavior. Naturally, which dependent measures are most appropriate depends on the problem being studied. For fault diagnosis, these might involve the system's ability at infer the correct qualitative diagnosis, its ability to explain future network behaviors, and the time take to detect and diagnose problems. Similar measures seem appropriate for responding to intruders and worms, though these might also include the speed and effectiveness of response. For studies of configuration, the dependent variables might concern the time taken to configure a new system and the resulting quality, which may itself require additional metrics. Similarly, routing studies would focus on the efficiency and effectiveness of the selected routes.

Note that these behavioral measures have nothing directly to do with learning; they are the same measures one would use to evaluate a nonlearning system and even the abilities of a human network manager. Because learning is defined as improvement in performance, we can only measure the effectiveness of learning in terms of the performance it aims to improve. Note also that the metrics mentioned above are quite vague, and they must be made operational before they can be used in experimental evaluations. In doing so, it may seem natural to use variables associated with one's selected formulation of the learning problem, such as predictive accuracy for classification or received reward for action selection. We should resist this temptation and instead utilize variables that measure directly what is desired from a networking perspective.

An experimental study also requires the variation of one or more independent factors to determine their effects on behavior. In general, these can deal with:

- the effects of experience, such as the number of observations available to the learning system;
- the effects of data characteristics, such as the degree of noise or percentage of features missing;
- the effects of task characteristics, such as the complexity of a configuration problem or the number of simultaneous faults;
- the effects of system characteristics, such as the inclusion of specific learning modules or sensitivity to parameter settings; and
- the effects of background knowledge, such as information about network structure and bandwidth.

Again, which variables are appropriate will depend largely on the networking problem at hand and the specific learning methods being used. However, a full understanding of how machine learning can assist cognitive networking will require studies that examine each of the dimensions above.

Of course, one cannot carry out experiments in the abstract. They require specific domains and problems that arise within them. To study the role of learning in network management, we need a number of testbeds that can foster the experimental evaluation of alternative approaches to learning. At least some of these should involve actual network, to ensure the collection of realistic data for training and testing the learning methods. However, these should be complemented with simulated networks, which have the advantage of letting one systematically vary characteristics of the performance task, the learning task, and the available data. Langley (1996) has argued that experiments with both natural and synthetic data are essential, since the former ensures relevance and the latter let one infer source of power and underlying causes.

Much of the success of the last 15 years of machine learning research can be traced to the establishment of a collection of data sets at the University of California, Irvine (Murphy & Aha, 1994, Blake & Merz, 1998; Merz & Murphy, 1996). The UCI data sets provided a common set of problems on which to evaluate learning algorithms and greatly encouraged comparative studies. The data sets span a wide range of application problems ranging from basic science and medicine to optical character recognition and speech recognition.

Ideally, we want an analog of this repository to enable the careful evaluation of machine learning in networking domains. However, because the Knowledge Plane envisions an adaptive network that learns about itself over time, it is important that this resource not be limited to static data sets, but also include simulated networks that allow learning methods, and their associated performance elements, to interact with the network environment in an on-line manner.

Bibliography

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37.

Baluja, S., & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. *The Proceedings of the 12th Annual Conference on Machine Learning* (pp. 38–46). San Francisco, CA: Morgan Kaufmann.

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *Proc. 11th Annu. Conf. on Comput. Learning Theory* (pp. 92–100). ACM Press, New York, NY.

Boyan, J., & Moore, A. (2000). Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research, 1*, 77–112.

Boyan, J. A., & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in Neural Information Processing Systems* (pp. 671–678). Morgan Kaufmann Publishers, Inc.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering, 8*, 195–210.

Cheeseman, P., Self, M., Kelly, J., Taylor, W., Freeman, D., & Stutz, J. (1988). Bayesian classification. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 607–611). Cambridge, MA: AAAI Press/MIT Press.

Clark, D. (2002). *A new vision for network architecture* (Technical Report). MIT.

Clark, P., & Niblett, T. (1988). The cn2 induction algorithm. *Machine Learning, 3*, 261.

Cypher, A. (1993). *Watch what I do: Programming by demonstration*. Cambridge, MA: MIT Press.

Drastal, G., Meunier, R., & Raatz, S. (1989). Error correction in constructive induction. *Proceedings of the Sixth International Workshop on Machine Learning (ML 1989), Cornell University, Ithaca, New York, USA, June 26-27, 1989* (pp. 81–83). San Francisco: Morgan Kaufmann.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187–226.

Jaffar, J., & Maher, M. J. (1994). Constraint logic programming: A survey. *Journal of Logic Programming, 19/20*, 503–581.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Kibler, D., & Langley, P. (1988). Machine learning as an experimental science. *Proceedings of the Third European Working Session on Learning* (pp. 81–92). Glasgow: Pittman.

Kumar, V. (1992). Algorithms for constraints satisfaction problems: A survey. *The AI Magazine, 13*, 32–44.

Langley, P. (1995). *Elements of machine learning*. San Francisco: Morgan Kaufmann.

Langley, P. (1996). Relevance and insight in experimental studies. *IEEE Expert*, 11–12. 20

Langley, P. (1999). User modeling in adaptive interfaces. *Proceedings of the Seventh International Conference on User Modeling* (pp. 357–370). New York: Springer.

Langley, P., & Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the ACM*, *38*, 55–64.

Merz, C. J., & Murphy, P. M. (1996). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Mitchell, T. M., Mahadevan, S., & Steinberg, L. I. (1985). LEAP: A learning apprentice for VLSI design. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 573–580). Los Angeles, CA: Morgan Kaufmann.

Moore, A., Schneider, J., Boyan, J., & Lee, M. S. (1998). Q2: Memory-based active learning for optimizing noisy continuous functions. *Proceedings of the Fifteenth International Conference of Machine Learning* (pp. 386–394). 340 Pine Street, 6th Fl., San Francisco, CA 94104: Morgan Kaufmann.

Moriarty, D. E., Schultz, A. C., & Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, *11*, 241–276.

Murphy, P., & Aha, D. (1994). *UCI repository of machine learning databases [machine-readable data repository]* (Technical Report). University of California, Irvine.

Myers, R. H., & Montgomery, D. C. (1995). *Response surface methodology: Process and product optimization using designed experiments*. New York, NY: John Wiley & Sons, Inc.

Partridge, C. (2003). *Thoughts on the structure of the knowledge plane* (Technical Report). BBN, Cambridge, MA.

Priebe, C. E., & Marchette, D. J. (1993). Adaptive mixture density estimation. *Pattern Recognition*, *26*, 771–785.

Quinlan, J. R. (1993). *C4.5: Programs for empirical learning*. San Francisco, CA: Morgan Kaufmann.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland and the PDP research group. (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, volume 1: Foundations*. Cambridge, MA: MIT Press.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. *Proceedings of the Ninth International Conference on Machine Learning* (pp. 385–393). San Francisco: Morgan Kaufmann.

Selman, B., Kautz, H. A., & Cohen, B. (1993). Local search strategies for satisfiability testing. *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*. Providence RI.

Sleeman, D., Langley, P., & Mitchell, T. (1982). Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, *3*, 48–52.

Stolcke, A., & Omohundro, S. (1994). Inducing probabilistic grammars by bayesian model merging. *Grammatical Inference and Applications: Proceedings of the Second International Colloquium on Grammatical Inference* (pp. 106–118). Springer Verlag.

Sutton, R., & Barto, A. G. (1998). *Introduction to reinforcement learning*. Cambridge, MA: MIT Press.

Tonge, F. M. (1963). Summary of a heuristic line balancing procedure. In E. A. Feigenbaum and J. Feldman (Eds.), *Computers and thought*, 168–190. Menlo Park, CA: AAAI Press/MIT Press.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229.

Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *1995 international joint conference on artificial intelligence*, 1114–1120. San Francisco, CA: Morgan Kaufmann.

Zweben, M., Daun, B., & Deale, M. (1994). Scheduling and rescheduling with iterative repair. In M. Zweben and M. S. Fox (Eds.), *Intelligent scheduling*, chapter 8, 241–255. San Francisco, CA: Morgan Kaufmann.