Published in the AAAI-94 Workshop on Case-Based Reasoning

A Comparative Utility Analysis of Case-Based Reasoning and Control-Rule Learning Systems

Anthony G. Francis, Jr. and **Ashwin Ram**
College of Computing, Georgia Institute of Technology
Atlanta, Georgia 30332-0280
{centaur, ashwin}@cc.gatech.edu

Abstract

The utility problem in learning systems occurs when knowledge learned in an attempt to improve a system's performance degrades performance instead. We present a methodology for the analysis of the utility problem which uses computational models of problem solving systems to isolate the root causes of a utility problem, to detect the threshold conditions under which the problem will arise, and to design strategies to eliminate it. We present models of case-based reasoning and control-rule learning systems and compare them with respect to the swamping utility problem. Our analysis suggests that CBR systems are more resistant to the utility problem than CRL systems.[1]

1. Introduction

All intelligent systems that learn can suffer from the utility problem, which occurs when knowledge learned in an attempt to improve a system's performance degrades performance instead (MINTON 1990). In this paper, we analyze the utility problem and examine its effects in case-based reasoners and control-rule problem solvers. Our methodology for this analysis couples a functional analysis of an AI system with a performance analysis of the system's algorithmic and implementational components. Such a computational model allows the identification of the root causes of the utility problem, which are combinations of algorithmic characteristics of an AI system (e.g., serial search of memory) with particular "parameters" that affect its operation (e.g., knowledge base size). Identifying the precise algorithmic nature of a root cause allows us to predict the threshold conditions under which it will affect a system severely enough to cause a utility problem.

Our analysis provides a general and theoretical framework for addressing this problem in systems that have been studied empirically (e.g., control-rule learning (CRL) systems) and in systems for which little utility analysis has been performed (e.g., case-based reasoning (CBR) systems). We use this framework to compare CBR and CRL systems, and find that CBR systems are more resistant to the utility problem than CRL systems.

2. Analyzing the Utility Problem

The utility problem was first detected in PRODIGY/EBL (MINTON 1988). PRODIGY/EBL is a *control-rule learner*,[2] a type of system that attempts to improve its problem-solving performance by learning search-control knowledge, called *control rules*, that reduce the amount of search it needs to perform by eliminating dead-end paths and selecting profitable ones. What Minton and others noticed about systems like PRODIGY/EBL was that the system could actually get slower after having learned control rules, rather than faster. At each step in the search space, a control-rule problem solver has to match all of its control rules against the current state to determine if they should fire. As that library of control rules grows in size, the cost of matching the control rules increases to the point that it outweighs or "swamps" the savings in search they provide.

This side effect of learning was called the "utility problem": learning designed to improve the system's performance ended up degrading performance instead (HOLDER ET AL. 1990). Since Minton's discovery of this "swamping" utility problem in PRODIGY, researchers have identified many different types of utility problems, each manifesting itself in slightly different ways. Because some types of utility problems are affected by the hardware architecture of the system and others are largely independent of hardware concerns, we can group the different types of utility problems into two rough classes: *architectural utility problems*, which arise from interactions between a system's learning and its hardware architecture, and *search-space utility problems*, which arise from interactions between learning and problem solving algorithms. A full discussion of the different types of utility problems is contained in (FRANCIS & RAM 1994). In this paper, we will focus on swamping, which is the utility problem most commonly encountered in learning systems. We will reserve the term "the utility problem" for the general utility problem, and will refer to specific versions of the utility problem, such as swamping, by their names.

3. A Methodology for Utility Analysis

We propose the use of algorithmic complexity theory as a tool for the analysis of the general utility problem. Our methodology involves analyzing different types of AI systems and decomposing their cognitive architectures into lower-level functional units, including problem-solving engines and memory systems, that can be repre-

[1]This research was supported by the United States Air Force Laboratory Graduate Fellowship Program and the Georgia Institute of Technology. A longer version of this paper (FRANCIS & RAM 1994) is ftp-able from `ftp.cc.gatech.edu:/pub/ai/ram/git-cc-94-24.ps.Z`

[2] Etzioni (1992) uses the term meta-level problem solvers for control-rule learning systems. We have avoided this term because of the possible confusion with metacognition, which includes systems that "know what they know" (metaknowledge) and systems that reason about their own reasoning processes (metareasoning, or introspection).

sented by formal algorithmic models. Our algorithmic approach incorporates both functional-level aspects of the computation, such as the system's cognitive architecture and its knowledge base, and implementation-level aspects, such as the performance characteristics of the system's hardware architecture. This multi-level analysis is crucial for the study of the utility problem because many utility problems arise due to interactions between the functional level of the system and the way that functional computation is actually implemented. Our methodology can be used to identify potential utility problems as well as to design coping strategies to eliminate their effects. In this paper, we focus on a comparative analysis of case-based reasoning and control-rule systems.

3.1. AI Systems and Learning

Formally, we can describe an AI system as a triple (**CA**, **KB, HA**). The *cognitive architecture* **CA** specifies a system in terms of separate functional modules that carry out fixed subtasks in the system, while the *knowledge base* **KB** represents the internal "data" that the **CA** uses to perform its computations. The *hardware architecture* **HA** defines the operations that a system can perform at the implementation level, as well as their relative costs. The cost (and hence the utility) of an operation may be different on different **HA**'s: for example, retrieval might take longer on a serial machine than it would on a parallel machine.

3.2. Utility and the Utility Problem

Utility can only be defined in terms of "performance" measures that judge the efficiency of a reasoner, such as execution time, number of states searched, storage space used, or even quality of solution. These *evaluation metrics* measure the costs that a system incurs during its reasoning. Given a particular evaluation metric, the *utility* of a learned item can be defined as the change in expectation values of a problem solver's performance on the metric across a problem set (MARKOVITCH & SCOTT 1993). To compute the utility of a change to the system's knowledge base with respect to some metric, we want to compute the costs that the system will incur for different problems weighted by the probability that the system will actually encounter those problems. Thus, utility is a function not only of the learned item but also of the learning system, the problem set, problem distribution, and the evaluation metric. The *utility problem* occurs when a learning system makes a change to its **KB** with the goal of improving problem solving utility on some metric by a calculated improvement F_c, but which has the side effect of degrading problem solving utility for another (possibly identical) metric by some actual amount F_a that outweighs the savings (i.e., $F_c<F_a$).

3.3. Dissecting the Utility Problem

In general, utility problems are not global, emergent properties of computation but can instead be tied to specific interactions between the **CA**, the **KB**, and the performance characteristics of the **HA**. In a CRL system, the interaction of interest is the relationship between match time and knowledge base size; in a CBR system, a similar interaction exists between case retrieval time and case library size.

We can formally define an *interaction* to be a combination of a set of parameters, a module, and a set of effects. The *module* represents the part of the **CA** that is responsible for the relationship between the parameters and their effects. *Parameters* represent characteristics of the system's knowledge base, while *effects* represent the performance measures that affected by the interaction. Thus, an interaction defines a function between learning (changes in the knowledge base) and performance (changes in the evaluation metric), mediated by the characteristics of the algorithmic component of the interaction (the module).

Utility problems arise when a learning module in the system causes parameter changes which interact with some **CA** component to produce "side" effects that impact the performance measures a learning module is designed to improve. This kind of coupling between a learning module and an interaction is a potential *root cause* of a utility problem. For a particular root cause, the calculated improvement F_c is the savings that the learning module is designed to perform, while the actual cost F_a is the actual change in performance taking into account the side effects of the interaction.

By comparing the algorithmic behavior of the learning module, the root cause interaction it is paired with, and the cost and savings functions that they contribute, we can compute *threshold conditions* — limiting values for the parameter changes that the system can tolerate before the actual costs exceed the calculated improvement and the system encounters a utility problem. Eliminating the general utility problem involves identifying the root causes of particular utility problems that can arise in a system and designing *coping strategies* that prevent their threshold conditions from being satisfied.

4. Modeling CRL and CBR Systems

To compare CBR and CRL systems, we must develop computational models of these systems and compare learning and problem solving in each.

The baseline for comparison of the computational model approach is the *unguided problem solver*. Unguided problem solvers (UgPS) are a class of problem solvers that operate without search control knowledge. The algorithm of a UgPS is a knowledge-free weak

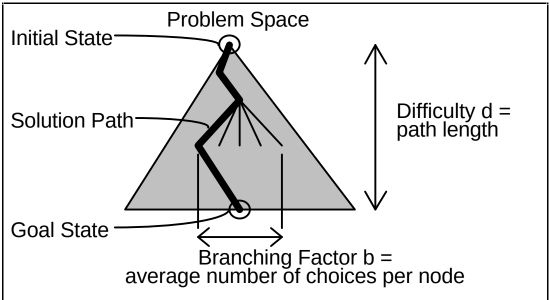
Problem Space
Initial State
Solution Path
Goal State
Difficulty d = path length
Branching Factor b = average number of choices per node

Figure 1. Unguided Search

method guaranteed to find a solution if one exists, such as breadth-first search; the only knowledge it uses is its *operator library*. For a given problem *p* with a solution of path length *d* operating in a search space with branching factor *b*, a UgPS will examine b^d nodes during search (see Figure 1). The number of nodes that the system expands is termed the *complexity* of a problem and is denoted C_p. Because the UgPS solves problems in exponential time, it can serve as a “baseline” against which more efficient learning systems can be compared. Much of “intelligence” can be viewed as attempts to reduce this combinatorial explosion through the use of heuristics or other techniques (NEWELL & SIMON 1975; RAM & HUNTER 1992; SCHANK & ABELSON 1977).

4.1. Control-Rule Problem Solvers

Learning systems improve over the UgPS by finding ways to reduce or eliminate search. A CRL system reduces search by retrieving and applying *control rules* at each state it visits during problem solving, giving it the ability to select or reject states. This control knowledge is a completely different kind of knowledge than operator knowledge and must be stored in a separate *control rule library*. If a system's control rule library is empty and

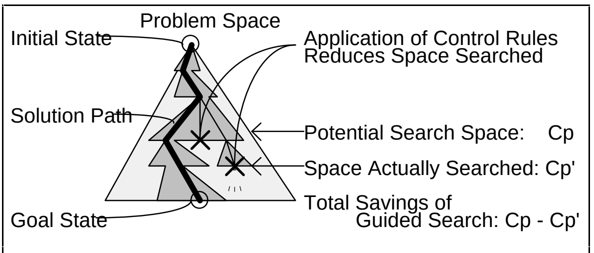
Problem Space
Initial State
Application of Control Rules Reduces Space Searched
Solution Path
Potential Search Space: Cp
Space Actually Searched: Cp'
Total Savings of Guided Search: Cp - Cp'
Goal State

Figure 2. Search Guided by Control Rules

control rules are not available, the problem solver resorts to blind search. Once a solution path has been found, correct decisions can be cached in the library as control rules that will guide the system in the future (see Figure 2). This model, while simplified, approximates many existing systems, including Soar and Prodigy.

4.2. Case-Based Reasoners

Case-based reasoning is primarily experience-based; when a CBR system encounters a new problem, it checks its *case library* of past problem solving episodes, or *cases*, looking for a similar case that it can adapt to meet the needs of the new problem. Our model of CBR has two primary knowledge libraries: the case library itself, indexed so that the most appropriate case can be retrieved in new problem-solving situations, and an *adaptation library* that stores *adaptation operators* that are used to transform the cases once they are retrieved. When a CBR system is presented with a problem, it retrieves an appropriate past case based on the problem's features, its goals, and the indices it has in its case library. Then, the case is adapted by performing search in the space of problem paths: the adaptation operators transform entire paths into new paths until a satisfactory

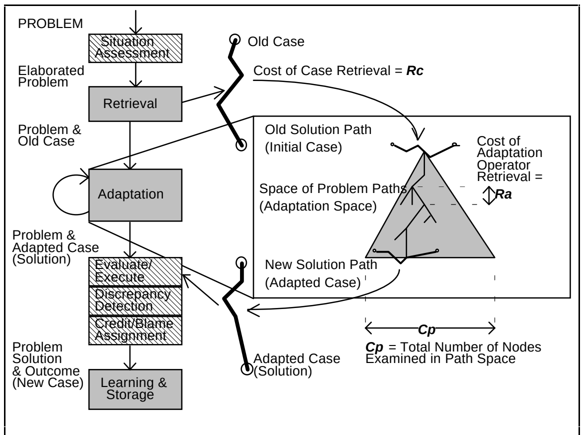
PROBLEM
Situation Assessment
Elaborated Problem
Retrieval
Problem & Old Case
Adaptation
Problem & Adapted Case (Solution)
Evaluate/ Execute
Discrepancy Detection
Credit/Blame Assignment
Problem Solution & Outcome (New Case)
Learning & Storage
Old Case
Cost of Case Retrieval = Rc
Old Solution Path (Initial Case)
Space of Problem Paths (Adaptation Space)
New Solution Path (Adapted Case)
Cost of Adaptation Operator Retrieval = Ra
Adapted Case (Solution)
Cp
Cp = Total Number of Nodes Examined in Path Space

Figure 3. Search in the Space of Problem Paths

solution path is achieved (see Figure 3). Once the new solution is found, it is stored in the case library indexed for future retrieval.

5. Analyzing Retrieval Costs

Our analysis of these models focuses on retrieval costs in CRL and CBR systems—how many retrievals are made, and how much does each retrieval cost? Retrieval is often cited as the core source of power for CBR systems, yet the retrieval cost is a critical factor in the swamping utility problem. An analysis of retrieval costs in CRL and CBR systems before and after learning reveals interesting differences in how each deals with retrieval.

For our analysis, we will assume that both the CRL and CBR systems operate on the same problem set, and that their problem spaces are defined by the same operator library. We further assume that the costs of adaptation operators are roughly equivalent to those of regular operators; this assumption may or may not be true for particular systems but is reasonable for this analysis.

We define a basic operation of retrieval, *R*, which chooses an item in a knowledge library based on some matching function. In general, for a given hardware architecture **HA**, the cost of retrieval for a knowledge library *i*, denoted R_i, is a function of both the library *i* and the item to be retrieved, *r*: $R_i = f(r,i)$. For a serial hardware architecture, $\mathbf{HA_s}$, the most important variable in this cost function is the number of items in the knowledge library, K_i. We will approximate this serial cost function with $R_i = cK_i$, where *c* is a constant multiplier that approximates the (nearly) linear cost function for matching on serial systems like $\mathbf{Ha_s}$.[3]

The particular interaction we will examine is the *retrieval time interaction*: the relationship between the retrieval operation R_i, knowledge library size K_i, and running time *t*. Because the learning operations in CBR and CRL systems have the effect of increasing the size of knowledge libraries in the system, their learning modules coupled with the retrieval time interaction form *po-*

[3]An indexed or parallel memory system might improve on this retrieval function (or, then again, might not, depending on the domain; see DOORENBOS 1993 and TAMBE ET AL. 1990).

tential root causes of the utility problem. Now, let's examine the dynamics of learning and retrieval in CBR and CRL systems and attempt to establish the *threshold conditions* for the utility problem in each.

5.1. Retrieval in Control-Rule Problem Solvers

In its initial state, without control rules, a CRL system is equivalent to the UgPS. It searches C_p states, retrieving a set of operators at each step with a cost of R_o. No control rules exist, so the cost of control-rule retrieval $R_c = 0$. Thus, the total cost in retrievals of the initial system is C_pR_o. After the system has learned a set of control rules, it has the capacity to guide its search. The number of states searched is reduced to C_p', where $C_p' < C_p$. However, in addition to retrieving a set of operators, it also needs to retrieve control rules at each step; thus, the cost for solving a problem rises to $C_p'(R_o+R_c')$.

The expected savings that guided problem solving brings are the costs of the states that the problem solver avoids: $(C_p - C_p')R_o$. The added costs are the costs of matching the control rules at each step: $C_p'(R_c' - R_c) = C_p'\Delta R_c$. (Note: because $R_c = 0$, $\Delta R_c = R_c'$). The utility problem will arise when the added costs exceed the expected savings. Thus, the threshold condition is $(C_p - C_p')R_o < C_p'\Delta R_c$; in other words, the swamping utility problem arises when the added cost of retrieval outweighs the benefits of individual rules. But will this threshold condition ever be met?

In the limit, the maximum search reduction is to a single path ($C_p' = d$), and operator retrieval costs are constant ($R_o' = R_o$) since the library of operators the system uses does not change in size. The *maximum expected savings* possible for any problem are thus $(C_p - d)R_o$. In contrast, the cost of retrieving control rules (R_c') increases without bound as the control base K_c increases in size; in the limit, the added costs associated with a rulebase are $dR_c' = dc(K_c)$ and thus can outweigh the maximum possible savings. Therefore, the threshold conditions *can* be met and the CRL system will encounter the utility problem.

These results indicate that swamping is a function of the potential speedup of learned items, the cost function of retrieval (which is itself dependent on retrieval strategies and machine architecture), and the number of items a system needs to learn. If the system converges on a bounded set of learned items and the hardware slowdown never approaches the utility of those items, the system will never be swamped.[4] If the learned items are of low utility, or if the learner never converges on a bounded set, as might be the case for an open-world or multidomain system, then the swamping problem can eliminate the benefits of the learned rules.

5.2. Retrieval in Case-Based Reasoners

To analyze utility effects in CBR systems, we need to measure the performance of a CBR system as it learns. To provide a basis for this measurement, we assume that a CBR system that does not have an appropriate case in memory can resort to some method (e.g., adaptation of a "null case" or "from-scratch" reasoning) that is cost-equivalent to the UgPS. Many existing CBR systems have such a last-resort method (e.g., KOTON 1989, KOLODNER & SIMPSON 1988).

A CBR system that resorts to null-case adaptation beginning with no experiences must still incur the cost of retrieving the null case (R_c) and then search the space of problem paths until the case has been adapted into a satisfactory solution. Under our earlier assumptions, the total number of paths the system examines is C_p, and one adaptation retrieval (R_a) occurs per step. Thus, the total cost of case adaptation before learning is $R_c + C_pR_a$. After the system has learned a library of cases, it will still need to retrieve a case but each case will require much less adaptation, reducing the number of paths examined to C_p' where $C_p' << C_p$. The cost of retrieving cases will increase to R_c' where $R_c' > R_c$. Thus, the total costs are $R_c' + C_p'R_a$.

To evaluate these results we must again examine the benefits and costs of case retrieval. The expected savings are the costs of the states that the problem solver avoids: $(C_p - C_p')\ R_a$, while the added costs are the increased costs of retrieval of cases $R_c' - R_c = \Delta R_c$. In the limit, the cost of retrieval increases without bound as the case library increases in size: $R_c' = c(K_c)$. However, as we approach the limit the case library contains many appropriate cases and little adaptation needs to be done—perhaps only one or two steps. In general, whenever the threshold condition $(C_p - C_p')\ R_a < \Delta R_c$ is met, the cost of retrieval outweighs the benefits of case adaptation; under these conditions, CBR systems will be swamped.

5.3. Advantages of Case-Based Reasoning

While this analysis reveals that both control-rule learners and CBR systems can suffer from the swamping utility problem, it also reveals that CBR systems have important advantages over CRL systems.

The primary advantage CBR systems have over control-rule problem solvers is that cases are retrieved only once during the lifetime of problem solving. For a control rule problem solver to avoid swamping, the increase in cost of retrieval of a control rule must be less than the fraction of total states that the system avoids in guided problem solving times the cost of an operator: $\Delta R_c < R_o(C_p - C_p')/C_p'$. For a CBR system, on the other hand, the increase in cost of a case retrieval must be less than the cost of the number of adaptation steps avoided: $\Delta R_c < R_a(C_p - C_p')$. The missing C_p' term in the denominator of the CBR equation arises because the increased cost of retrieval of control rules are incurred at each step in the search space, whereas the increased cost of case retrieval

---

[4] For example, on a parallel machine with a logarithmic cost function $R_i = c(\log K_i)$, the threshold condition $(C_p\text{-}C_p')R_o < C_p'c(\log K_c)$ may never be met in a closed-world domain in which a small set of knowledge items learned by rote are adequate for performance. If the learning system successfully converges on a small enough set, the logarithmic slowdown will be negligible compared to the potential savings. This condition can arise on serial architectures as well, but because the cost function is linear in the size of the knowledge base the constraints on the size of the learned set are much more severe.

is incurred only once during problem solving for a CBR system. In other words, CBR systems *amortize* the cost of case retrieval across all adaptations, making them much more resistant to increases in retrieval costs than CRL systems.

This amortization also makes CBR more amenable to solutions to the swamping problem, such as deletion policies or indexing schemes. In order to be effective, any coping strategy needs to reduce retrieval time to the point that the threshold conditions are never satisfied. For CRL systems, this upper limit is $R_c' < R_o(C_p - C_p')/C_p'$; for CBR systems, this upper limit is $R_c' < R_a(C_p - C_p')$, a much higher (and hence much less stringent) limit on the maximum time retrieval can take for a system to be guaranteed to avoid swamping.

5.4. Future Comparative Analysis

The above analysis does not close the book on the utility problem in CBR and CRL systems. There are a number of other differences that can contribute to the utility problem, including differences in operator costs, degree of search reduction, and knowledge base size.

Normal problem solving operators and adaptation operators do not necessarily have comparable costs. A traditional problem solving operator makes a change to a single state, while an adaptation operator can potentially make several changes to a case. Thus, the same reduction in C_p' in CBR and CRL systems could provide different degrees of savings because the individual steps being saved may not cost the same.

Another difference between CBR and CRL systems is that they are not guaranteed to produce the same reduction in C_p' after being exposed to the same set of training examples. While both types of systems are senstitive to the structure of the domain, search reduction in CRL systems depends on the learning and matching policy for control rules, but in CBR systems depends on the indexing scheme and similarity metric used for case retrieval. Therefore, the same training set might produce different search reductions for CRL and CBR systems.

Furthermore, depending on the learning biases chosen, different systems could extract varying numbers of cases or rules from the same set of examples. Therefore, even if the search reduction is the same, there is no guarantee that the size of the systems' knowledge libraries will be comparable, and hence no guarantee that their performance on the utility problem will be identical.

Providing a principled account of the effect contributing factors like these have on the utility problem is the primary goal of our research. While we believe we have identified one factor — amortization — which contributes to the performance differences of CBR and CRL systems, it is only one contributing factor among many that determine the systems' respective performances. Accounting for other contributing factors, such as operator costs, degree of search reduction, and knowledge base size, will allow a principled comparison of CBR and CRL systems performance on the utility problem.

For example, there have been few attempts to systematically evaluate the cost-utility tradeoffs in CBR systems with very large case libraries. However, reports of the speedup provided by cases (e.g., KOTON 1989) and control rules (e.g., TAMBE ET AL 1990) suggest that cases can provide large improvements, up to an order of magnitude greater than the speedups provided by control rules. It is *possible* that the utility of cases may be high enough to allow CBR to avoid swamping, but it is not clear whether this will always be the case.

6. The Bottom Line

CBR's patterns of retrieval make it resistant to the utility problem. Because the cost of case retrieval is amortized over many adaptation steps, ideal CBR systems suffer less severely from the same overhead and are more amenable to coping strategies than CRL systems. This analysis suggests several future lines of comparative research on CBR and CRL, such as operator costs, degree of search reduction and size of knowledge bases.

References

Doorenbos, R.B. (1993) Matching 100,000 Learned Rules. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, p290-296. AAAI Press/MIT Press.

Etzioni, O. (1992) An Asymptotic Analysis of Speedup Learning. In *Machine Learning: Proceedings of the 9th International Workshop*, 1992.

Francis, A. and Ram, A. (1994). Computational Models of the Utility Problem and their Application to a Utility Analysis of Case-Based Reasoning. *Technical Report GIT-CC-94-24*, College of Computing, Georgia Institute of Technology.

Holder, L.B.; Porter, B.W.; Mooney, R.J. (1990). The general utility problem in machine learning. In *Machine Learning: Proceedings of the Seventh International Conference*, 1990.

Kolodner, J.L. & Simpson, R.L. (1988). The Mediator: A case study of a case-based reasoner. Georgia Institute of Technology, School of Information and Computer Science, Technical Report no. GIT-ICS-88/11. Atlanta, Georgia.

Kolodner, J.L. (1993). *Case-based Reasoning.* Morgan Kaufmann, 1993.

Koton, P. A. (1989). A Method for Improving the Efficiency of Model-Based Reasoning Systems. Laboratory for Computer Science, MIT, Cambridge, MA. Hemisphere Publishing.

Markovitch, S. and Scott, P.D. (1993) Information Filtering: Selection Methods in Learning Systems. *Machine Learning*, 10: 113-151.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3), March 1990.

Newell, A. & Simon, H.A. (1975). Computer Science as Empirical Inquiry: Symbols and Search. Reprinted in Haugeland, J., (ed.). *Mind Design: Philosophy, Psychology, Artificial Intelligence*, chapter 1, pp 35-66. MIT Press, 1981.

Ram, A. & Hunter, L. (1992) The Use of Explicit Goals for Knowledge to Guide Inference and Learning. *Applied Intelligence*, 2(1):47-73.

Schank, R. & Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*. LEA, Hillsdale, NJ.

Tambe, M.; Newell, A.; Rosenbloom, P. S. (1990). The Problem of Expensive Chunks and its Solution by Restricting Expressiveness. Machine Learning, 5:299-348, 1990.