Published in *Machine Learning: ECML-95: 8th European Conference on Machine Learning, Heraclion, Crete, Greece, April 1995. Proceedings. Lavrac, N. & Wrobel, S. (eds). Lecture Notes in Artificial Intelligence, vol 912. Springer-Verlag.*

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
(404) 853-9381, (404) 853-9372
{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 utility problems 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 their performance with respect to the swamping utility problem. Our analysis suggests that case-based reasoning systems are more resistant to the utility problem than control-rule learning systems.[1]

1. Introduction

An interesting asymmetry exists in the patterns of retrieval in case-based reasoning (CBR) and control-rule learning (CRL) systems: to take advantage of past learning experiences, CRL systems need to retrieve rules from memory at each step, whereas CBR systems need retrieve a case only once. Under certain conditions, this asymmetry may provide CBR with an advantage in dealing with the *utility problem*, which arises when knowledge learned in an attempt to improve a system's performance degrades performance instead (HOLDER ET AL. 1990, MINTON 1990). In this paper, we analyze the differences between CBR and CRL systems in the context of a general methodology for the study of the utility problem. Our methodology couples a functional analysis of a problem solving system with a performance analysis of the system's algorithmic and implementational components. This computational model allows us to formally specify the root causes of the utility problem in terms of interactions within the system and to predict the threshold conditions under which the utility problem will arise. Using this methodology, we have found that, while both CBR and CRL systems can suffer from the utility problem, CBR systems have important advantages over CRL systems. In particular, because CBR systems amortize the cost of case retrieval over many adaptation steps, ideal case-based reasoners suffer less severely from the same overhead than CRL systems.

[1]This research was supported by the United States Air Force Laboratory Graduate Fellowship Program and the Georgia Institute of Technology.

2. Analyzing the Utility Problem

2.1. What is the Utility Problem?

The utility problem was first detected in PRODIGY/EBL (MINTON 1988). PRODIGY/EBL is a control-rule learning system,[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 CRL system 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 often increases to the point that they outweigh the savings in search the rules provide.

This side effect of learning was called the “utility problem”: learning designed to improve the system's performance ended up degrading performance instead. Since Minton's discovery, 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 and search-space utility problems (FRANCIS & RAM 1993).

Architectural utility problems arise when learning has the side effect of causing an increase in the costs of basic operations the system performs; for example, in PRODIGY/EBL learning new control rules caused the cost of retrieval to rise. Two types of architectural utility problem have been identified: *swamping*, which arises in systems like PRODIGY/EBL when the cost of matching a large number of rules “swamps” the savings (MINTON 1990); and *expensive chunks*, which arises when a few individual rules are so expensive to match that they outweigh the benefits of the rest (e.g., TAMBE ET AL. 1990). *Search-space utility problems* arise because of the manner in which learning modifies the search performed by a problem solver, and not because of limitations in the system's underlying hardware architecture. Three have been identified in CRL systems: *branching*, which arises when a system learns macro-operators that increase the branching factor of the search space (ETZIONI 1992), *wandering*, which arises when a learner fails to achieve tractability (ETZIONI 1992), and *composability*, which arises when learned control rules interfere with each other (GRATCH & DEJONG 1991).

A full discussion of the different types of utility problems is beyond the scope of this paper; we will focus on swamping, which is the utility problem most commonly encountered in learning systems. In this paper, we will reserve the term “the utility

[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).

problem” for the general utility problem, and will refer to specific versions of the utility problem, such as swamping, by their names.

2.2. The Methodology

We propose the use of algorithmic complexity theory as a tool for the analysis of the 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 represented 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.

For a comparative analysis to be successful, the AI systems being studied must be modeled with a uniform vocabulary of basic cognitive operations that is sufficient to describe the architectures of a wide range of systems. This uniform representational language will allow us to represent AI systems as *computational models* whose basic operations are identical and thus are suitable for comparative algorithmic complexity analysis. The performance of different systems can then be directly compared in terms of the costs of basic cognitive operations, such as memory retrieval or operator applications. These cognitive operations can in turn be modeled on different hardware architectures to determine their costs and the utility of learning in these systems. Our methodology can therefore be used to identify potential utility problems, as well as to design coping strategies to eliminate their effects.

3. A Quick Introduction to Utility Analysis

3.1. AI Systems and Learning

Formally, we can describe an AI system as a triple (**CA**, **KB, HA**) of cognitive architecture, knowledge base, and hardware architecture. 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 types of operations that a system can perform at the implementation level, as well as the relative costs of such operations. 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.

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. Because many utility problems are dependent on the distribution of problems that a system encounters (e.g., TAMBE ET

AL. 1990), we must also represent the *problem set* **PS**, which is defined by a tuple (**S, D**) of a *problem space* **S,** the space of problems a system can encounter, and a *problem distribution* **D,** the probability that the system will encounter a particular problem.

3.2. Utility and the Utility Problem

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). In other words, when we 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, and the evaluation metric. The *utility problem* occurs when a learning system makes a change to its knowledge base **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) evaluation 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 cognitive architecture, the knowledge base and the performance characteristics of the hardware architecture. 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 independent variables in the interaction (the parameters) and the dependent variables (the 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 cognitive architecture 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

The baseline for comparison of the computational model approach is the *unguided problem solver*. Unguided problem solvers use knowledge-free weak methods, and are always guaranteed to find a solution if one exists; one such method is breadth-first search. Given a problem *p* whose solution is a path of length *d*—which we shall call the *depth* or *difficulty* of the problem—an unguided problem solver will expand on the average b^d nodes during its search, where *b* is the branching factor of the search space. The number of nodes that the system expands for a problem *p* is termed the *complexity* of a problem and is denoted C_p. Figure 1 depicts the search space of an unguided problem solver.

The only knowledge library that an unguided problem solver uses is its *operator library*; its algorithm consists of applying whatever operators are allowable in all of the possible sequences that begin with its starting point. The unguided problem solver serves as a "baseline" against which learning systems can be compared because it is the "worst" system, in terms of performance, that is capable of solving a particular problem in a given search space. An unguided problem solver solves problems in exponential time in the size of the problem; 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; SIMON 1993).

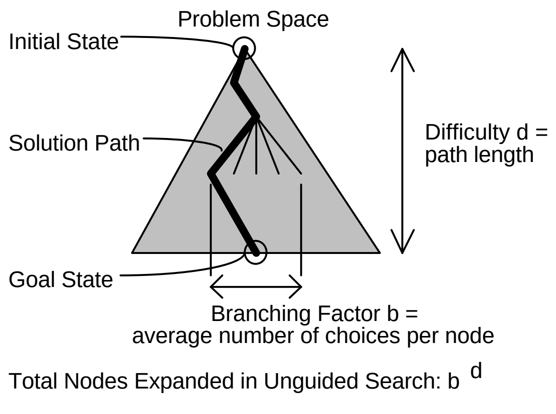
Problem Space
Initial State
Solution Path
Difficulty d = path length
Goal State
Branching Factor b = average number of choices per node
Total Nodes Expanded in Unguided Search: b^d

Figure 1. Unguided Search

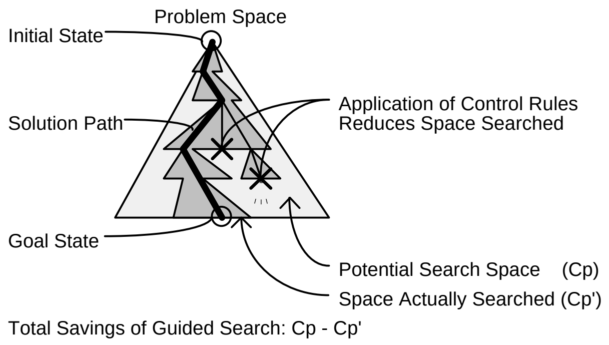
Problem Space
Initial State
Solution Path
Application of Control Rules Reduces Space Searched
Goal State
Potential Search Space (Cp)
Space Actually Searched (Cp')
Total Savings of Guided Search: Cp - Cp'

Figure 2. Search Guided by Control Rules

4.1. Control-Rule Learning Systems

Learning systems improve over the unguided problem solver model by finding ways to reduce or eliminate search. A control-rule learning 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 control rules are not available, the problem solver resorts to blind search. Once a solution path has been found, the correct decisions can be cached in the library as control rules that will guide the problem solver in similar situations in the future. This model, while simplified, is a good approximation of many existing systems, including Soar and Prodigy. Figure 2 depicts the guided search of a CRL system.

4.2. Case-Based Reasoners

Case-based reasoning is primarily experience-based; when a case-based reasoner 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[3] 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 case-based reasoner 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. Once a case has been retrieved, the case is adapted by performing search in the space of problem paths: the

[3]This model leaves out some of the stages of a full-fledged case-based reasoner, such as situation assessment and credit/blame assignment (KOLODNER 1993), but contains the core “case-based” elements of the full case-based reasoning approach.

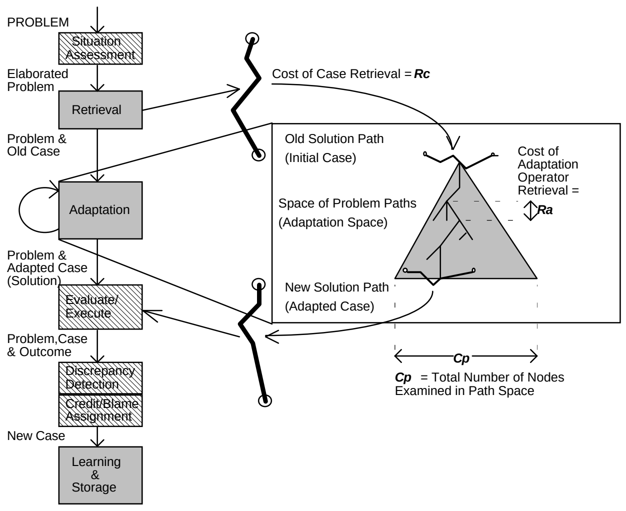
PROBLEM
Situation Assessment
Elaborated Problem
Retrieval
Problem & Old Case
Adaptation
Problem & Adapted Case (Solution)
Evaluate/ Execute
Problem,Case & Outcome
Discrepancy Detection
Credit/Blame Assignment
New Case
Learning & Storage
Cost of Case Retrieval = Rc
Old Solution Path (Initial Case)
Space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

Figure 3. Search in the Space of Problem Paths

adaptation operators are used to transform entire paths into new paths until a satisfactory solution path is achieved. Figure 3 depicts this search in the space of problem paths. Once the new solution is found, it is stored in the case library, indexed by the goals of the current problem for future retrieval.

5. Analyzing Retrieval Costs

To illustrate how computational models can be used to analyze utility effects in different kinds of problem solving systems, consider retrieval costs in CRL and CBR systems—how many retrievals are made, and how much does each of those retrievals cost? Retrieval is often cited as the core source of power for CBR systems, yet the cost of retrieval is a critical factor in the swamping utility problem. An examination of retrieval costs, both before and after learning, in CRL and CBR systems reveals both the source of the swamping problem and potential mechanisms for its solution.

Because the focus of this comparison is on the differences in retrieval between CBR and CRL systems, we will make the simplifying assumption that "all other things are held equal." Specifically, we will assume that both the CRL and CBR systems operate on the same problem set, and moreover that the problem space they function in is defined by the same operator library.

We define a basic operation of retrieval, R, which extracts an item (such as a case or operator) from 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: $R_i = f(r,i)$. Some of the features of a knowledge library that affect retrieval time are the number and organization of items in the system's knowledge library. However, 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 $\mathbf{HA_S}$. For the purposes of this paper, we will ignore the possible variations in the match cost of each individual item, as well as other issues dependent on more specifics of the cognitive architecture of the memory system and the hardware architecture upon which it runs.

Because the learning operations in both case-based reasoners and control-rule learning systems have the effect of increasing the size of knowledge libraries in the system, the learning modules in both types of systems, combined with the retrieval time interactions in each, form potential root causes of the utility problem. The particular interaction we will examine, therefore is the retrieval time interaction: the relationship between the system parameters R_i (number of retrieval operations) and K_i (knowledge library size), and the effect t (running time). Given these definitions, let us examine the actual dynamics of learning and retrieval in CRL and CBR systems and attempt to establish the threshold conditions for the utility problem in each.

5.1. Retrieval in CRL systems

In its initial state, without control rules, a CRL system is equivalent to an unguided problem solver. It searches C_p states, retrieving a set of operators at each step with a cost of R_o. 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 CRL brings are the costs of the states that the problem solver avoids, or just $(C_p - C_p')R_o$. The added costs are the costs of matching the control rules at each step, $C_p'R_c$. Obviously, the utility problem will arise when the added costs exceed the expected savings. Thus, the threshold condition is $(C_p - C_p')R_o < C_p'R_c$; in other words, when the cost of retrieval outweighs the benefits of individual rules. This is, in effect, the swamping utility problem. But will this threshold condition ever be met? To determine this, we must examine how large the expected savings in states can become, and how that compares to the added costs of retrieval.

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 case-based reasoning 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 using first principles problem solving to produce a sketchy case which can then be adapted), and this method is no worse than an unguided problem solver. Most existing CBR systems have such a last-resort method; for example, the earliest case-based reasoner, MEDIATOR, had a rule-based problem solving method that it could fall back on if no case was available (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 from the library but each case will require much less adaptation, reducing the number of paths examined to C_p' where $C_p' << C_p$. Also, the cost of retrieving cases may 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 casebase increases in size: $R_c' = c(K_c)$. However, as we approach the limit the casebase 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.

[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.)

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.

One advantage of CBR systems is that they have a greater potential improvement than CRL systems. Even if a CRL system learns enough rules to guide search completely, with no false paths, the control rules and operators must nevertheless be retrieved and applied at each step; this means the minimum cost of solution in a CRL system will be dR_oR_c. In contrast, if a case that completely solves the current problem is retrieved from the case library, no adaptation will need to be done and the minimum cost will be just R_c. In practice, it is just as unrealistic to assume that the case library will have an exact case as it is to assume that a CRL system will be able to completely guide search. Some adaptation will need to be done, for a total cost of $C_p'R_a + R_c$. The precise tradeoffs between case-based and CRL systems depend on the particular domain, but nevertheless the *potential* savings are greater for CBR systems.

Another advantage CBR systems have over CRL systems is that cases are retrieved only once during the lifetime of problem solving. For a CRL system 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

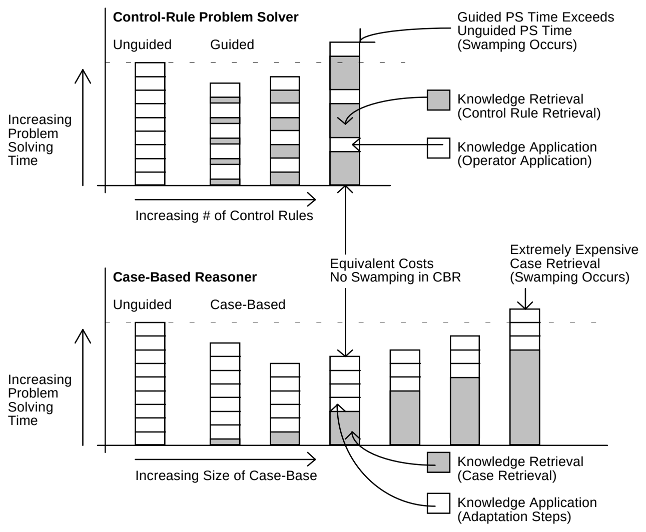
Control-Rule Problem Solver
Unguided
Guided
Guided PS Time Exceeds Unguided PS Time (Swamping Occurs)
Knowledge Retrieval (Control Rule Retrieval)
Knowledge Application (Operator Application)
Increasing Problem Solving Time
Increasing # of Control Rules
Equivalent Costs No Swamping in CBR
Extremely Expensive Case Retrieval (Swamping Occurs)
Case-Based Reasoner
Unguided
Case-Based
Increasing Problem Solving Time
Increasing Size of Case-Base
Knowledge Retrieval (Case Retrieval)
Knowledge Application (Adaptation Steps)

Figure 4. Comparing Control Rules and Cases

of case retrieval is incurred only once during problem solving for a case-based reasoner. 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.

Figure 4 illustrates this phenomenon. Each bar graph represents the time a problem solver takes to solve a problem plotted against its knowledge base size; white blocks in the bar graph represent problem solving or adaptation steps, while grey blocks represent knowledge retrievals. Because a CRL system makes a retrieval from memory at each step, the costs of retrieval (illustrated by the size of the grey blocks) can outweigh the savings of reduced search more quickly than they can in a case-based reasoner, in which retrieval from a large knowledge library is (ideally) performed only once during problem solving.

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 a CRL system, this upper limit on retrieval time is $R_c < R_o(C_p\text{-}C_p')/C_p'$; for a CBR system, this upper limit is $R_c < R_a(C_p\text{-}C_p')$, 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.

Whether the benefits suggested by this analysis will be realized in practice depends critically on other factors that may affect the cost-benefit relationships within the system. Specifically, this analysis claims that, for CBR and CRL systems operating over the same problem set and same operator library, an identical increase in retrieval costs is less harmful to the case-based system than it is to the CRL system. However, the actual retrieval costs of the two systems may be different over the same learning history; moreover, the two types of systems may differ in the amount of improvement they gain from learning. Therefore, while this analysis *suggests* that there may be advantages for CBR systems, actually determining the tradeoffs between particular CBR and CRL systems requires a more precise specification of the costs and specifics of the learning algorithms of the systems, as well as of the characteristics of the problem set over which the systems will operate.

6. Related Work

Little research has been done on direct comparisons of case-based reasoning and control-rule learning systems. Minton (1990) and Etzioni (1992) have theoretically and empirically investigated the utility of control-rule learning systems, and Koton (1989) empirically evaluates the utility of a particular case-based reasoning system against a non-learning reasoner.

Our theoretical model of control-rule learning systems was based in part on Etzioni's theoretical model of control-rule learning systems, which he calls meta-level problem solvers (ETZIONI 1992). While we developed our theoretical model of case-based reasoning systems independently, it shares many characteristics with the Systematic Plan Adaptor (SPA) algorithm (HANKS & WELD 1995), a case-based reasoning system built on top of an unguided partial-order planning system called SNLP (BARRETT & WELD 1994).

A closely related body of work concerns DerSNLP (IHRIG & KAMBHAMPATI 1994). DerSNLP is a derivational analogy system also layered on top of SNLP. Ihrig and Kambhampati use the DerSNLP framework to theoretically and empirically evaluate the efficiency of plan-space and state-space planning systems. While their work and ours both focus on comparative theoretical analysis of AI systems, their work compares two different planning algorithms in the context of the same learning algorithm and our work compares two different learning algorithms in the context of the same planning system.

7. The Bottom Line

The utility problem is caused by interactions between a system's learning modules and other portions of a system and can arise completely independently of hardware concerns. Analyzing the utility problem at an algorithmic level allows us to identify the root causes responsible for the problem and to identify the threshold conditions under which the problem will arise; solving the utility problem involves applying coping strategies which prevent these threshold conditions from occurring. Our uniform framework for utility analysis also facilitates the direct comparison of different systems on the utility issues. We are currently extending this framework to account for more factors that can affect the utility problem, such as domain boundedness, representation language and indexing vocabulary.

Several features of CBR make it resistant to the utility problem. First, cases have the potential to eliminate vast amounts of problem solving, providing improvements robust enough to survive large increases in retrieval time. Second, because the cost of case retrieval is amortized over many adaptation steps, ideal case-based reasoners suffer less severely from the same overhead than CRL systems. Finally, this amortization makes CBR systems more amenable to coping strategies than CRL systems.

References

Barret, A. & Weld, D. (1994). Partial order planning: evaluating possible efficiency gains. *Artificial Intelligence, 67* (1), 71-112.

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

Francis, A. and Ram, A. (1993). Computational Models of the Utility Problem and their Application to a Utility Analysis of Case-Based Reasoning. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning,* pages 48-55, University of Massachusetts at Amherst, June 30, 1993.

Gratch, J.M. and Dejong, G.F. (1991) Trouble with gestalts: The composability problem in control learning. Technical Report, University of Illinois at Urbana-Champaign, April 1991.

Hanks, S. & Weld, D.S. (1995). A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research* 2 (1995), 319-360.

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.

Ihrig, L. & Kambhampati, S. (1994). On the Relative Utility of Plan-Space v. State-Space Planning in a Case-Based Framework, Technical Report 94-006, Department of Computer Science and Engineering, 1994. Arizona State University.

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, 1989.

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,* Morgan Kaufmann, 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*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Simon, H.A. (1993). Artificial Intelligence as an Experimental Science. Invited Talk. Abstracted in *Proceedings of the Eleventh National Conference on Artificial Intelligence*, page 853, July 11-15, 1993.

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.