To appear in the Proceedings of the Workshop on Knowledge Compilation and Speedup Learning (KCSL 93), June 30, 1993

Computational Models of the Utility Problem and their Application to a Utility Analysis of Case-Based Reasoning

Anthony G. Francis, Jr.
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
centaur@cc.gatech.edu

Ashwin Ram
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
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. There are many types of utility problems and many factors that combine to cause them; computational models of problem solving systems which take these factors into account can be used to analyze the causes of the utility problem and to design coping strategies to eliminate it. The application of this analysis to case-based reasoning reveals that it may suffer from several utility problems, and suggests coping strategies that may be effective in combating these problems.

1. INTRODUCTION

All AI 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 (HOLDER ET AL. 1990, MINTON 1990). There are many types of utility problems and many factors that combine to cause it; to thoroughly analyze the utility problem we must develop computational models of problem solving systems which allow us to identify the root causes of the utility problem. Our analysis reveals that the utility problem, while exacerbated by the costs of current serial computers, is not an artifact of machine architecture alone; it is also affected by the type and scope of the problem solving system, the characteristics of the problem domains and the representational language used to model the domain, and the types of learning performed. To avoid the utility problem, then, we must explicitly analyze the costs arising from the different root causes of the problem in various types of problem solving systems and design coping strategies to eliminate these costs.

In this paper, we identify the factors involved in different types of utility problems and propose a method for the computational analysis of these factors. We use this method to analyze different types of problem solving systems, including unguided search, control rule, and case-based systems. While some aspects of the utility problem have been studied in some kinds of problem solvers, our analysis provides a general and theoretical framework for addressing this problem in problem solvers that have been studied empirically (e.g., control-rule learning systems) and in problem solvers for which little utility analysis has been performed (e.g., case-based reasoning systems). In particular, our analysis reveals that case-based reasoning systems may suffer from several different utility problems, and suggests coping strategies that would help avoid these problems as these systems are scaled up.

2. WHAT IS THE UTILITY PROBLEM?

In general, the utility problem occurs when learning new knowledge hurts system performance. This simple characterization, however, does not capture the full richness of the problem. In practice, the utility problem appears in many forms, and many factors interact in subtle ways to cause it. For example, the utility problem is affected by the architecture of the learning system, the problem solving method, the types of problems the system encounters, the types of representations the system uses, and even the hardware architecture that the problem solving system runs on.

Some types of utility problems are affected by the hardware architecture of the system, particularly the degree of parallelism; others are largely independent of hardware considerations. Therefore, it is useful to group the different types of utility problems into two rough classes based on this distinction: *architectural utility problems* and *search-space utility problems.*

2.1. ARCHITECTURAL UTILITY PROBLEMS

Architectural utility problems arise when learning has the side-effect of causing the costs of cognitive operations that

the system performs to rise. On traditional serial computers, architectural utility problems most often arise because of the rising cost of memory access and matching as the knowledge base (or the items it contains) increases in size. Learning algorithms that do not take this overhead into account can cause a system to slow down more than the average speedup provided by individual learned rules. There are two primary kinds of architectural utility problems: swamping and expensive chunks.

Swamping was first identified in PRODIGY/EBL (MINTON 1990). Swamping occurs when a system learns a large number of rules whose combined match time slows problem solving time down more than individual control rules speed it up. Each individual rule may be useful and easy to match, but the overhead of matching the entire rule set outweighs any performance benefit provided by the individual rules. Swamping is, in a sense, the "core" utility problem because of the O(*n*) cost of retrieval in serial computers. For this same reason, it is the most amenable to solution by a move to parallelism.

The *expensive chunks* problem was first discovered in the Soar system (TAMBE ET AL. 1990). The problem occurs when a few individual control rules are so expensive to match that they slow down the entire system. The overhead of the rest of the knowledge base may not cause a slowdown at all. Many algorithms for matching control rules are NP-complete operations, and can become arbitrarily expensive if rule size is unbounded. The performance effect of expensive chunks is independent of the size of the database, and can have serious repercussions *despite* a move to parallel hardware. Current solutions include restricting the expressiveness of matching items to prevent match time from rising without bound, but this solution often has the side effect of increasing the number of items the system needs to learn, which may return us to the swamping problem.

2.2. SEARCH-SPACE UTILITY PROBLEMS

In contrast to architectural utility problems, search-space utility problems arise due to the manner in which learning modifies the search performed by a problem solver, and not because of a slowdown of the system's underlying cognitive architecture. Such problems are in general a side effect of the types of items that an algorithm learns and can only be eliminated by careful design of learning algorithms. There are three major classes of search-space problems: branching, wandering, and composability.

The *branching* utility problem can arise when a problem solver learns macro-operators that change the topology of the search space itself. If the macro-operators increase the branching factor of the search space even slightly, the additional number of nodes visited can increase work and search time in the limit despite the improvement provided by the macro-operators (ETZIONI 1992).

The *wandering* utility problem occurs when a set of control rules fails to curtail search in the search space to a polynomial function of the problem difficulty. While the set of rules may provide improvement on problems of the same difficulty for which they are learned, problem solving time guided by the ruleset may still be exponential in solution depth, a situation which is generally regarded to be intractable (ETZIONI 1992).

Finally, the *composability* problem occurs when individual control rules that improve problem solving performance actually degrade problem solving performance when used in conjunction (GRATCH & DEJONG 1991A). While each rule may provide some calculated speedup, in conjunction they provide no speedup at all.

Solving these utility problems requires a careful analysis of the type of knowledge the system learns. For the composability problem in particular, this is extremely difficult because control rules can theoretically interact in many different ways. In control-rule problem solvers, search-space utility problems are best resolved by never allowing them to occur in the first place (GRATCH AND DEJONG 1991B); however, this still does not guarantee that the system will avoid architectural utility problems.

3. WHAT CAUSES THE UTILITY PROBLEM?

Each of the different types of the utility problem has a major symptom—for example, the rise in match time in swamping or expensive chunks—and an apparent cause for that symptom—for swamping, the size of the knowledge base, and for expensive chunks, the size of the items that a memory system learns. But these visible causes are themselves merely symptomatic of deeper relationships between the problem solving system, the types of problems that it encounters, and the machine architecture upon which it runs. These deeper factors are the true causes of the utility problem, and a thorough account of them is necessary to fully understand the problem.

For example, one factor that can affect the utility problem is the difficulty of the problems that the system faces; the more difficult the problems, the more likely it is that a system will encounter the utility problem. However, the difficulty of a problem is not a simple function but can be broken down into a number of factors, including the boundedness, dynamicism and size of the problem domain, the size of the problems the system may encounter, the representation language in which problems are expressed, and even the scope of the problem solving system itself.

One of the more interesting distinctions in problem-

solving theory, and one that captures precisely the distinction between real world problems and several artificial problem domains such as the blocks world, is the degree of *boundedness* of the domain, better known as the open/closed world distinction. A *closed world* is any domain in which a problem solver's knowledge can be complete and consistent (HINRICHS 1992). Examples include tic-tac-toe and the abstract problem of chess. An *open world* is any domain in which a problem solver's knowledge is not complete or consistent. Examples include Szechwan cooking and international diplomacy. Some ways that a domain can be open include *open class categories* (categories to which items can be added at any time, such as nouns and rivers), *incomplete domain theories* (e.g., cooking or biology), or *underspecified problems* (e.g., meal planning). In the real world, of course, the actual practice of even a "simple" game such as chess is effectively an open world problem, because the limited resources available to a reasoner makes reasoning from domain principles intractable.

Closely related to the boundedness of a domain is its *dynamicism*—how much the domain changes over time. In a static world, the features of the world or domain do not change with time. Examples include blocks worlds domains. Dynamic domains, in contrast, change with time. Dynamic domains are usually open, although in a few special cases the changing world may not invalidate the problem solver's knowledge.

The *size of the domain*—how much knowledge is required to represent the domain itself—is also important. However, "small" and "large" domains are relative terms, and can be defined only in terms of a particular system's capacity. The more knowledge is required to represent the domain, the more complex the problems that the system encounters will be and the greater the branching factor of the search space. A good contrast is a small blocks world vs. medical diagnosis.

The *size of the problems* that the systems solves is also important. Small problems have relatively low search depths and/or small branching factors, requiring expanding a number of nodes in search that is relatively small compared to the capacity of the reasoner. Large problems have deep search trees or wide branching factors, or, alternatively, have interactions between rules that complicate search for states. The game of go is an example of a a small, static closed domain in which large problems can arise. Like domain size, problem size can only be defined relative to a particular problem solver.

All of these factors are affected by the *representation language* used to represent the problem and the domain. The representation language can affect both the complexity of the domain and the size and difficulty of solving problems in that domain (e.g., AMAREL 1966) as well as the difficulty and effectiveness of indexing (e.g., KOLODNER 1993). As such, the choice of representation language for a system is a critical factor in whether and to what extent the utility problem will arise. For example, reducing the expressiveness of a representation language can eliminate the expensive chunks problem (TAMBE ET AL. 1990), but it brings the risk of introducing the swamping problem.

The *scope of the system* also affects the difficulty of the problems it encounters and the difficulty in building a system that is competent to solve them. Systems range in scope from single domain systems to multidomain systems. *Single-domain* systems operate with only one domain and usually only one type of problem. A problem solving system limited to a single domain can make simplifying assumptions that ease the problem solving process; this is the process of *taking advantage of ignorance* (LENAT & GUHA 1989).

In contrast, *multidomain* systems operate in multiple domains; that is, the same system is capable of solving different problems given to it in different domains. This is not the same thing as reconfiguring a general-purpose AI system to operate in more than one domain by reloading a new domain theory. Problem solvers working simultaneously in multiple domains must have more flexible knowledge representation and processing capacities; in this respect, there are many shared features between multidomain systems and systems that function in open worlds. Case-based and analogical reasoning systems that use knowledge in one domain to solve problems in another also share many features with true multidomain systems. Cyc is an example of a multidomain system (LENAT & GUHA 1989).

Other factors that can affect the utility problem are the *type of the problem solver* itself, such as rule-based, case-based, and model-based; the *type of learning* that a system performs, such as rote learning, control-rule learning, case learning; and the *hardware architecture* upon which the problem solver runs. These factors are considered in more detail in the analysis below.

4. MODELING THE UTILITY PROBLEM

4.1. THE METHODOLOGY

Most research into the utility problem has focused on the problem's symptomatic features, such as the match cost of an entire rulebase (MINTON 1988) or the match cost of individual rules (TAMBE ET AL. 1990). Analyses tend to focus on single variables, such as the problem domain (TAMBE ET AL. 1992), and solutions tend to focus on techniques that alleviate the symptoms, such as applying restricted expressiveness to eliminate reduce the match cost of expensive chunks (TAMBE ET AL. 1990). While

many of these solutions have been shown to be effective empirically, the lack of attention to the interactions between the solutions and the underlying factors that cause the utility problem leaves entirely open the question of whether these techniques are general cures for the utility problem for all types of problem solvers in all types of domains. We need a principled analysis that takes into account all of the possible causes of the utility problem in order to determine which factors cause which utility problems on what kinds of problem solvers.

We propose the use of algorithmic complexity theory to analyze the cognitive architectures of AI systems with respect to problem solving performance and utility issues in various types of problem domains. This analysis results in computational models of cognitive architectures which can be used to identify potential utility problems as well as to design coping strategies to eliminate their effects. We have used the approach to extend Etzioni's (1992) analysis of unguided and control-rule problem solvers,[1] and also to model case-based reasoners and rote learners.

Our methodology involves analyzing the cognitive architectures of different types of AI systems, including high-level systems like problem solvers and lower-level building blocks like memories, and developing a representational vocabulary of basic cognitive operations that is sufficient to describe the systems. This uniform representational language allows us to represent different AI systems as *computational models* suitable for computational complexity analysis. While a full examination of the models we have developed is beyond the scope of this paper, some of the issues we have identified in modeling these systems include determining what libraries of knowledge the system uses, what state information it uses, how these knowledge sources are used in the system's problem-solving algorithm, when learning occurs, what type of items the system learns, how often it learns, and how much those learning operations cost.

The uniform representational language allows us to model all of these aspects of problem solving systems in the same vocabulary and to directly compare different systems' performance in terms of the costs of basic cognitive operations, such as memory storage, memory retrieval, or operator applications. The cognitive operations themselves can in turn be simulated on different machine architectures to determine the total time and space costs.

4.2. EXAMPLES OF COMPUTATIONAL MODELS

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.[2] Given a problem p whose solution is a path of length d—which we shall call the *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. Learning systems improve over the unguided problem solver model by finding ways to reduce or eliminate search —that is, they reduce the complexity C_p for problems of a given difficulty d.

The utility problem has primarily been studied in a class of speedup learning systems called meta-level or *control-rule problem solvers* (ETZIONI 1992). These systems augment the unguided problem solver model by the application of control rules at each problem step, which allows the system to select or reject states. If 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 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.

Case-based reasoning incorporates many techniques that are designed to improve its performance, from the use of cases themselves to avoid search to special-purpose heuristics that guide the adaptation of cases. However, there is little theoretical justification for the case-based reasoning paradigm: no principled examination of case-based reasoning has been conducted, and few case-based reasoning systems have been scaled up far enough to evaluate the utility aspects of such systems.

We are developing a computational model to evaluate utility issues in case-based reasoning. Under this model, past problem solving episodes, or *cases*, are stored in a *case library*, indexed so that the most appropriate case is *retrieved* in new problem-solving situations. Once a case has been retrieved, case-based reasoners *adapt* the case by performing search in the space of problem spaces, using

---

[1] Etzioni 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).

[2] Of course, such an ideal weak method may not actually exist for a particular domain. This is the issue of competence — determining what knowledge and processing structures a system needs to merely function in a domain, much less improve its performance. While we consider competence issues to be extremely important in the choice of coping strategies and learning systems, we are shelving this issue for this paper for two reasons. First, even competent systems can suffer from the utility problem. Second, competence issues are likely to make the utility problem *worse*, not better. Therefore, the simplifying assumptions we are making about system competence are actually conservative, and while considering competence issues could serve to reinforce our results it would introduce major complexities that are beyond the scope of this paper.

adaptation operators to transform entire paths into new paths until a satisfactory solution path is achieved. Once the new solution is found, it is *indexed* based on the goals of the problem, and is *stored* in the case library for future retrieval.

While this simplified model leaves out many possible extensions of case-based reasoning, such as predicting possible failures, learning from outcomes, and learning adaptation strategies, it does provide a reasonable approximation for the purposes of our analysis. For example, we can see that a kind of expensive chunks problem might arise when the cost of computing and/or matching expensive indices outweighs the savings of applying a case, or when the adaptation cost of case outweighs savings in search. Swamping can occur when the costs of matching indices against a large casebase to find the best case outweighs the savings of retrieved cases. A kind of branching problem may occur when the process of adaptation causes search in the problem space that are avoided in blind search, whereas wandering may occur when the benefits provided by cases becomes vanishingly small as case size increases. Finally, composability utility problems arise when individually good indices cause the system to ignore better cases when used in conjunction.

While no systematic study of these problems exists, some of these tradeoffs have been considered in the design of many existing case-based reasoning systems. Although some of these systems incorporate methods to avoid these problems, there is no guarantee that the solutions applied in these systems will scale to larger systems. Furthermore, because case-based reasoning's adaptation process searches the problem space in an entirely different way than traditional problem solvers—by altering whole paths, rather than extending a path towards a goal—it may encounter entirely new utility problems which have not yet been classified and for which no solutions are yet known.

5. ANALYZING RETRIEVAL COSTS

To illustrate how computational models can be used to analyze utility effects in different kinds of problem solving systems, consider the retrieval costs of the memory module —how many retrievals are made, and how much does each of those retrievals cost? The cost of retrieval is a critical factor in performance utility problems, and an examination of these costs, both before and after learning, in control rule problem solvers and case-based reasoners reveals both the source of the swamping problem and potential mechanisms for its solution.

For the purposes of this section, we will assume that retrieval occurs from a knowledge library based on some matching function. For example, a set of operators might be retrieved from a library of operators based on literals in the current state that match its preconditions, or a case might be retrieved from a case library based on similarities between the goals in the previous situation and the system's current goals. The cost of this retrieval, denoted R_i, is a function of both what is to be retrieved and the number and organization of items in the knowledge library. We will approximate this cost function with $R_i = c(K)$, where K is the number of items in the knowledge library and c is a function that approximates the linear cost increase of matching on serial systems. 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 the cognitive architecture of the memory system and the hardware architecture upon which it runs.

5.1. RETRIEVAL IN CONTROL-RULE PROBLEM SOLVERS

In its initial state, without control rules, a control-rule problem solver 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 savings that control-rule problem solving 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, $C_p'R_c$. But how large are the savings in states, and what are the actual costs of these retrievals?

In the limit, the maximum search reduction is to a single path ($Cp' = d$), 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 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 increases in size; in the limit, the added costs associated with a rulebase are dR_c and can outweigh the maximum possible savings. In general, whenever $(C_p-C_p')R_o < C_p'R_c$, the cost of retrieval outweighs the benefits of individual rules. This is, in effect, the swamping utility problem.

These results indicate that swamping is a function of the potential speedup of learned items, the cost function of the retrieval process (which is itself dependent on retrieval strategies and machine architecture), and the number of items that the system is expected 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. 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 utility problem can eliminate the benefits of the learned rules.

One method to avoid the problem would be to delete low-utility rules (MINTON 1990) or to put an upper limit on the size of the knowledge base and to delete rules if the limit is exceeded (SAMUELS 1959). These *deletion policies* might have detrimental effects on multidomain systems or systems that mix speedup and knowledge-level learning in the same format, such as Soar and most case-based reasoning systems. Another method might be to restrict the amount of the knowledge base searched on any one retrieval, perhaps by using indexing or some kind of quick matching to eliminate many learned items from consideration. In order to be effective, these kinds of *guided search policies* need to limit search to a subset of the knowledge base that is no larger than the subset that would be left by a deletion policy.

5.2. RETRIEVAL IN CASE-BASED REASONERS

To analyze utility effects in case-based reasoning (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 (for example, 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. We also assume that adaptation operators are as powerful as regular operators, that is, the system can always find the solution through adaptation (even from a null case) if one exists.

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_o) occurs per step. Thus, the total cost of case adaptation before learning is $R_c + C_p R_a$.

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[3] 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 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 $R_c' - R_c$. In the limit, the cost of retrieval increases without bound as the casebase increases in size; however, as we approach the limit the casebase contains many appropriate cases and very little adaptation needs to be done—perhaps only one or two steps. In general, whenever $(C_p - C_p')R_a < R_c' - R_c$, the cost of retrieval outweighs the benefits of case adaptation; under these conditions, CBR systems will face the swamping problem.

5.3. ADVANTAGES OF CBR SYSTEMS OVER CONTROL-RULE PROBLEM SOLVERS

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 control rule learners.

One advantage of CBR systems is that they have a greater potential improvement than control-rule problem solver. Even if a control-rule problem solver 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 for a control rule learner will be $dR_o R_c$. On the other hand, 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 control rule learner will be able to completely guide search. Some adaptation—perhaps only one or two steps—will need to be done, for a total cost of $C_p' R_a + R_c$. The precise tradeoffs between this cost and the cost of guided search depend on the particular domain, but nevertheless the *potential* savings are greater for case-based reasoning systems.

Another advantage CBR systems have over control-rule learners is that cases are retrieved only once during the lifetime of problem solving. For a control rule learner to avoid swamping, the 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: $R_c < R_o(C_p - C_p')/C_p'$. For a case learner, 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: $R_c < R_a(C_p - C_p')$. 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 is incurred only once during problem solving

[3] Of course, most CBR systems employ some kind of indexing scheme to optimize matching, so retrieval costs will not necessarily increase, depending on the domain and the indexing scheme. CBR systems in general incorporate a wide range of optimizing strategies that can improve performance, and one of the goals of this research is to provide a thorough account of how effective these strategies are at eliminating the utility problem.

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 control-rule problem solvers.

Thus, while the added costs of retrieval can still outweigh the maximum possible savings in CBR systems, the costs of case retrieval are amortized across all adaptations, and the potential savings are greater than those of control-rule problem solvers. Therefore, deletion or guided search policies to eliminate the swamping problem in case-based reasoning have less stringent requirements than similar strategies for control-rule problem solvers.

6. WILL THE UTILITY PROBLEM GO AWAY?

Most AI research and hence most of the study of the utility problem has been conducted on serial von Neumann architectures, where match cost grows linearly with the size of the knowledge base. Because of these limitations, it is often claimed that the utility problem is an artifact of serial architectures, and that massive parallelism will simply make the performance utility problem go away by reducing match cost to nearly constant time. Unfortunately, an examination of the utility problem reveals flaws with this analysis.

For example, as we have just demonstrated, the swamping utility problem is directly caused by the cost of retrieval R_i. Retrieval of memory items requires both matching of items in memory *and* selecting the best matching item. On serial systems both of these operations cost O(n), which leads to the high cost of retrieval in traditional serial computers. On large parallel systems, matching the knowledge library can be completely parallelized, but theoretical considerations of parallel architectures enforce a lower bound of $\Omega(\lg n)$ on the time complexity of ideal selection algorithms on physically realizable computers (COOK ET AL 1986, CORMEN ET AL. 1990, PARBERRY 1990). While this is certainly an improvement over an O(n) match time on serial systems, it can still give rise to the utility problem, especially in problem-solving systems that operate in open worlds, with large knowledge bases full of items that may give small amounts of improvement.

On the other hand, in closed world domains with small problems and sufficiently large parallel hardware, even a rote learner can avoid swamping because the total number of problems that the system encounters is small enough that an logarithmic slowdown will never be larger than the potential improvement. These types of complex interactions make the need for a principled analysis of the utility problem all the more pressing. In CBR systems, for example, while parallelism may eliminate much of the slowdown in matching costs, the utility problem may resurface in other guises, such as the increasing cost of computing effective indices for very large case bases. An adequate representational vocabulary for similarity judgements of cases—in other words, a content theory of index labels—can be critical in controlling this cost.

The other types of utility problem may be less amenable to parallel solutions. The expensive chunks problem is caused by the cost of matching individual memory items, which is an NP-complete problem that contains a large serial component (NORVIG 1989). The branching, wandering, and composability problems are more problematic; they are entirely independent of the hardware architecture of the system and can only be eliminated through the redesign of learning algorithms.

7. SO, HOW CAN WE COPE?

To eliminate the utility problem, we must identify its causes—rising hardware costs, unbounded match item size, rule interactions, and so on—and design *coping strategies* that either eliminate the causes or reduce their effects. For example, because the swamping utility problem arises from the cost of searching an entire knowledge base for matching items, coping strategies can limit the increase of that cost by limiting the amount of matching the system needs to do. Two coping strategies to solve this problem have already been mentioned: deletion policies and guided search policies. Other strategies include the use of *information filters* to restrict the harmfulness of knowledge (MARKOVITCH & SCOTT 1989), such as a selective acquisition filter that restricts learning to only extremely high utility items that will be resistant to the slowdown (COHEN 1990).

Earlier we showed that guided search strategies such as indexing need to restrict the search of the knowledge base to at least the degree that a deletion policy would in order to prevent the utility problem. The Rete pattern matcher (FORGY 1981) incorporates another form of guided search policy through its use of alpha and beta matching phases; however, its criteria for the alpha phase is determined purely by fast matching of the items and there is no *a priori* limit on the number of items that can be then fully matched and compared in the beta phase. This predicts that the Rete algorithm would be successful at eliminating the rise in match costs in some domains but not in others, an effect that has already been empirically demonstrated for the retrieval algorithms used in the Soar system, which are based on the Rete approach (TAMBE ET AL. 1992).

Coping strategies for the expensive chunks problem include restricting the expressiveness of matching items or placing an upper bound on the amount of time a match is permitted to proceed. Other types of coping strategies may include allowing matching to proceed asynchronously (allowing the memory system to return its current best match and to alert the cognitive module if it finds a better

match later) and guiding search in the knowledge base through the careful design of indices and search strategies.

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 an architectural slowdown. 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 conventional problem solvers. Finally, while CBR systems can suffer from the expensive chunks problem, they can easily incorporate a restricted expressiveness policy into the indexing scheme by placing an upper bound on the size of an item that can be matched (e.g., DOMESHEK 1992).

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) appear to show that cases can provide very large improvements over unguided problem solver performance, 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 systems to avoid the swamping utility problem altogether, but it is not at all intuitively clear that this will always be the case. The memory module in a CBR system, for example, must be designed such that $\Delta R_C < (C_p\text{-}C_{p'})R_a$—in other words, the increase in cost of a case retrieval must never be greater than the potential savings of adaptation.

8. THE BOTTOM LINE

The utility problem is a real problem and not simply an artifact of current hardware. It occurs on both serial and parallel machines, although it is much easier to cope with on parallel systems. It affects a wide range of problem-solving systems using a variety of learning strategies, including macro-operator systems, control-rule problem solvers, CBR systems, and perhaps all types of learning systems. It is sensitive to the type of problem domain as well; it rarely arises in simple domains where a small set of rules or cases is sufficient to provide complete coverage, but in problem-solvers operating in multiple domains or in open worlds it can cripple or eliminate the system's ability to speed up its performance through learning. Even a change in the representational language a system uses can affect whether or not it will suffer from the utility problem.

Solving the utility problem involves devising coping strategies that eliminate either the causes of the problems or provide relief from their effects. Case-based reasoning's success at speedup learning can be tied to its successful use of the coping strategies of amortization and its particular method of problem solving; together, these methods make CBR very resistant to the swamping utility problem as systems scale up. However, CBR systems may encounter a number of other utility problems, and additional techniques will need to be derived to prevent them.

Acknowledgements

This research was supported by the United States Air Force Laboratory Graduate Fellowship Program.

References

(AMAREL 1967) Amarel, S. "On Representations of Problems on Reasoning about Actions." Reprinted in Nilsson & Webber, eds., Readings in Artificial Intelligence, Tioga Press. 1967.

(COHEN 1990) Cohen, W.W. "Learning approximate control rules of high utility." In *Machine Learning: Proc.7th Int'l Conf.*, 1990.

(COOK ET AL. 1986) Cook, S.; Dwork, C.; Reischuk, R. "Upper and lower time bounds for parallel random access machines without simultaneous writes." *SIAM Jrnl on Computing*, 15(1):87-97, 1986.

(CORMEN ET AL. 1990) Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. *Introduction to Algorithms*. MIT Press, McGraw-Hill. 1990.

(DOMESHEK 1992) Domeshek, E. "Do the right thing: A component theory for indexing stories as social advice." *Tech. Report #26*, May 1992. Institute for the Learning Sciences, Northwestern University.

(ETZIONI 1992) Etzioni, O. "An Asymptotic Analysis of Speedup Learning." In *Machine Learning: Proc. 9th Int'l Workshop*, 1992.

(FORGY 1982) Forgy, C.L. "Rete: A fast algorithm for the many pattern/many object pattern match problem." *Artificial Intelligence* 19(1):17-37, 1982.

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

(GRATCH & DEJONG 1991B) Gratch, J.M. and Dejong, G.F. "Utility generalization and composability problems in explanation-based learning." *Technical Report*, University of Illinois at Urbana-Champaign, Illinois, August 1991.

(HINRICHS 1992) Hinrichs, T.R. *Problem Solving in Open Worlds: A Case Study in Design.* Lawrence Erlbaum, 1992.

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

(KOLODNER 1993) Kolodner, J.L. *Case-based Reasoning.* Morgan Kaufman, 1993. In press.

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

(LENAT & GUHA 1990) Lenat, D.B. and Guha, R.V. Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project. Addison-Wesley Publishing Company, Inc., 1990.

(MARKOVITCH & SCOTT 1989) Markovitch, S. and Scott, P.D. "Utilization Filtering: a method for reducing the inherent harmfulness of deductively learned knowledge." In *IJCAI-89: Proc. of the 11th International Joint Conference on Artificial Intelligence*, 1989.

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

(NORVIG 1989) Norvig, P. *Paradigms of AI Programming: Case Studies In Common Lisp*. Morgan Kaufman, 1993.

(PARBERRY 1987) Parberry, I. *Parallel Complexity Theory.* John Wiley & Sons, 1987.

(SAMUELS 1959) Samuels, A.L. "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal of Research and Development*, 3, 211-229.

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

(TAMBE ET AL. 1992) Tambe, M.; Doorenbos, R.; Newell, A. "The Match Cost of Adding a New Rule: A Clash of Views." *Technical Report CMU-CS-92-158*, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, June 1992.