*Abstracted in **Case-Based Reasoning: Papers from the 1993 Workshop**, July 11-12, Washington, D.C., Technical Report WS-93-01, AAAI Press*

THE UTILITY PROBLEM IN CASE BASED REASONING

ANTHONY G. FRANCIS
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 351-4574
centaur@cc.gatech.edu

ASHWIN RAM
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 853-9372
ashwin@cc.gatech.edu

ABSTRACT

Case-based reasoning systems may suffer from the utility problem, which occurs when knowledge learned in an attempt to improve a system's performance degrades performance instead. One of the primary causes of the utility problem is the slowdown of conventional memories as the number of stored items increases. Unrestricted learning algorithms can swamp their memory system, causing the system to slow down more than the average speedup provided by individual learned rules. Massive parallelism is often offered as a solution to this problem. However, most theoretical parallel models indicate that parallel solutions to the utility problem fail to scale up to large problem sizes, and hardware implementations across a wide class of machines and technologies back up these predictions. Failing the creation of an ideal concurrent-write parallel random access machine, the only solution to the utility problem lies in a number of coping strategies, such as restricting learning to extremely high utility items or restricting the amount of memory searched. Case-based reasoning provides an excellent framework for the implementation and testing of a wide range of methods and policies for coping with the utility problem.

KEYWORDS

Utility problem, parallelism, complexity theory, case-based reasoning

1. INTRODUCTION

The paradigm of case-based reasoning is based on the fundamental principle that it is advantageous to remember your past experiences and to re-apply them in novel situations. Case-based reasoning systems operate by remembering their experiences in problem solving episodes and recalling the solutions when they encounter new problems. Generalized traces of past problem-solving episodes, called *cases*, act as a cache of previous solutions, allowing systems to compute new solutions more quickly and efficiently (CARBONELL 1986, HAMMOND 1989, KOLODNER 1993). While many intuitive arguments can be made for the appeal of this approach, it is unclear that case-based reasoning will be exempt from many of the problems that plague conventional learning systems. In particular, case-based reasoning, like many other learning systems, may suffer from the utility problem.

The *utility problem* in learning systems occurs when knowledge learned in an attempt to improve a system's performance degrades it instead. The problem appears in many AI systems, but it is most familiar in speedup learning. Speedup learning systems are designed to improve their performance by learning control rules which guide their problem-solving performance. These systems often exhibit the undesirable property of actually slowing down if they are allowed to learn in an unrestricted fashion (MINTON 1990, HOLDER 1990). Each individual control rule is guaranteed to have a positive utility (improve performance) but, in concert, they have a negative utility (degrade performance).

One of the causes of the utility problem is the serial nature of current hardware. The more control rules that speedup learning systems acquire, the longer it takes for the system to test them on each cycle. Few case-based reasoning systems have been tested with case-bases large enough to encounter this kind of problem, so the utility problem has not been extensively studied. It is not intuitively clear that case-based reasoning is in any way exempt from the utility problem.

One solution to the utility problem is to design a parallel memory system to eliminate the increase in match cost (DOMESHEK 1992, KOLODNER 1988). This approach moves the matching problem away from the central processor and into the memory of the system. These so-called *active memories* allow memory search to occur in "nearly constant-time" in the number of data items (FAHLMAN 1988), relying on the memory for fast, simple inference and reminding.

It is not possible to simply store cases in a database structure indexed on some pre-computed hash key for easy retrieval. In general, matching cases requires performing a partial match between a case and features of a situation — an exponential-time problems in the worst case — and then selecting a case out of the matching cases. Unfortunately, while matching of multiple cases can be done in parallel, theoretical and practical considerations of parallel architectures enforce a lower bound of $\Omega(\lg n)$ on the time complexity of ideal selection algorithms.

This seemingly innocuous bound places in reality a set of harsh constraints on any retrieval algorithms designed to solve the utility problem. An analysis of these constraints in turn reveals that case-based reasoning, while incapable of solving the utility problem by itself, provides a ready framework for the implementation of a series of coping strategies that can.

2. PROBLEM-SOLVING AND LEARNING SYSTEMS

In order to evaluate case-based reasoning, it is important to have other kinds of systems to compare case-based reasoning systems with. Particularly important for this discussion are classes of learning and problem solving systems that suffer from the utility problem.

2.1. STATIC AND DYNAMIC LEARNING SYSTEMS

Not all systems suffer from the utility problem; for the purposes of this discussion, we can

divide learning systems into two categories; static and dynamic. Static machine learning systems, such as some category-learning and discovery algorithms, are more properly called *learning methods*, because the algorithms never modify themselves or the knowledge that they use. In contrast, *dynamic* learning systems, such as Soar (TAMBE ET.AL. 1990), CHEF (HAMMOND 1989), and PRODIGY (VELOSO & CARBONELL 1993), integrate learning and problem-solving performance. These programs modify the knowledge that they use during the course of learning and problem-solving.

The utility problem only arises in dynamic learning systems. Moreover, it primarily arises in systems that have a secondary goal of speeding up their processing on some task other than the learning task itself. Informally, these systems acquire knowledge which should improve their performance, and yet their performance degrades because of the additional processor overhead the new knowledge imposes.

This leads us to the question of defining the possible performance tasks that a system might engage in. Given such a definition, we can characterize the additional overhead that learning imposes on performance, and from that analyze and evaluate possible solutions to the utility problem.

2.2. A TAXONOMY OF PROBLEM-SOLVING SYSTEMS

Perhaps the most general characterization of perfomance tasks is as instances of *problem solving*. Given a set of *operators* that define a *problem space*, a *problem* can be defined as a pair of an *initial state* and a *goal state* in that problem space. Problem solving is the process of finding a path between the initial state and the goal state, called a *solution*. A *problem solver*, then, is simply an algorithm that performs this search process.

The length of the solution path is called the depth or *difficulty* of the solution, and is denoted by d. The amount of work, w, that a problem solver has to do to find a solution defined to be the number of nodes that the problem solver visits during the course of problem solving. The total problem solving time t that the system takes is simply the product of the work and the average *time cost* of each problem solving step, c:

$$t=cw$$

A final definition worth note is that of the *size* of a problem, s. A convenient definition of size is the number of bits needed to represent the problem; note, however, that the size of a problem description may have no bearing whatsoever on the difficulty of its solution. To see this, imagine a problem space defined by program states and possible program inputs, and a problem whose initial state is the initial state of a program and the goal state is that of a halted program. Since it is easy to devise short programs that halt immediately, halt after arbitrarily long running times, or never halt at all, it is correspondingly easy to devise three problems of the same size whose solutions have vastly different difficulties — the first (immediate halt) with a difficulty of 1, the second with a difficulty as great as we choose, and the third with an undefined difficulty (because it is not soluble at all).

2.2.1. UNGUIDED PROBLEM SOLVERS

An *unguided problem solver* is a problem-solving procedure that has no knowledge about which operators to apply at a given state, yet nonetheless is guaranteed to find a solution. An unguided problem solver is simply one of the standard weak methods of artificial intelligence. One such method is breadth-first search, which is the benchmark method we will use for comparison in this paper. For breadth-first search, the work that a problem solver must perform to find a solution to a problem is exponential in its depth:

$$w_{\varnothing} = b^d,$$

where *b* is the average *branching factor* of the search space. To distinguish the work and time that an unguided problem solver takes to solve a problem, these values are denoted $w_{\varnothing}$ and $t_{\varnothing}$; the $\varnothing$ symbolizes the lack of control knowledge — the "empty" knowledge base — used by the system.

2.2.2. META-LEVEL PROBLEM SOLVERS

A useful formalism for studying the utility problem is the *meta-level problem solver* model (ETZIONI 1992). Many existing systems that suffer from the utility problem can be modeled as meta-level problem solvers, including Soar and PRODIGY.

Meta-level problem solvers augment a standard unguided problem solver by using a database, *K*, of *control rules*, which guide the problem solver's choices of which states to visit. At each state that a meta-level problem solver visits in a problem space, the system must match its database against the current state to determine which control rules to apply. A meta-level problem solver with no control knowledge degenerates to an unguided problem solver. With very little modification, this definition of control rules can be extended to many kinds of knowledge, from standard control rules to chunks to certain kinds of macro-operators.

The work and time that a guided problem solver takes to solve a problem are denoted by w_K and t_K, where *K* represents the knowledge that the system uses to solve the problem.

2.2.3. CASE-BASED REASONING SYSTEMS

In this framework, case-based reasoning systems are dynamic learning systems that use a database of past solutions, called *cases*, to *avoid* search in the problem space. The basic case-based reasoning model uses a *problem solving engine* to produce the system's initial cases; for the purposes of this model we will assume that this is an unguided problem solver, although this need not be the case. The cases are then generalized and stored in a database, *K*, called a *case library*. On future problem solving episodes, a *retrieval engine* searches the case library for appropriate cases. A retrieved case is then *adapted* by the problem solving engine to produce the solution.

Actual case-based reasoning systems depart from this simple model in a number of ways. Most real case-based reasoning systems use some kind of guided problem solver to perform adaptation, either using generic control rules in addition to the operators or storing the equivalent of control rules with each case. Not all case-based reasoning systems can function without cases. In these systems, the problem solving engine is more appropriately termed an *adaptation engine*, because it cannot solve the problem on its own and merely serves to adapt the initial cases that the system has obtained from observation, instruction or preprogramming. Other case-based reasoning systems use cases to predict and avoid failures as well as to plan for successes. Still other systems can apply case-based reasoning recursively to perform adaptation; these systems more closely resemble meta-level problem solvers than the basic case-based reasoning model. For the purposes of the formal analyses in this paper, however, we will restrict ourselves to the basic case-based reasoning model.

2.3. RETRIEVAL IN META-LEVEL PROBLEM SOLVERS AND CASE-BASED REASONERS

For both meta-level problem solvers and case-based reasoners, there are two key operations that are crucial to the system's ability to guide or avoid search through the search space. The first operation is the *matching* of the knowledge base to the current state, to determine which elements of a knowledge base apply in the current situation. The second operation is the *selection* of a particular element to apply in the current situation.

The combined operations of matching and selection are collectively called *retrieval*. The cost of retrieval obviously depends on the number of items *n* in the knowledge base *K*, as well as on the algorithms for matching and selection. On serial systems, the increased cost of retrieval is a

key factor in the utility problem.

3. The Utility Problem as an Interaction of Performance Metrics

The utility problem, in its most general form, is the phenomenon whereby a learning system **L** acquires a knowledge base **K** with the goal of *improving* problem solving performance over unguided problem solving by some calculated factor f_C, but whose actual effect is to *degrade* problem solving performance by some actual factor f_a. In this context, better performance means that the system expends less labor when solving the same problems, given some metric *m* of labor like problem solving work or time. By this definition, the utility problem arises when the calculated improvement factor $f_C<1$, but the actual change in performance $f_a>1$.

It is sometimes convenient to refer to the calculated improvement factor f_C simply as the speedup, *F*, and to refer to the ratio of the actual to the calculated improvements f_a/f_C as the slowdown, *S*. Thus, the speedup can be represented by:

$$f_C = m_{projected}/m_{current}$$

and the actual improvement can be expressed by the formula:

$$f_a = Sf_C.$$

Most frequently, the utility problem arises when the problem solving system *calculates* its speedup with regards to one metric of system performance, such as problem solving work, but the system's performance is *measured* by a different metric, such as problem solving time (Tambe et.al. 1990). For example, both case-based reasoners and meta-level problem solvers decrease the number of states that the system visits during problem solving, but this improvement may be offset by the increased overhead *O* imposed by the process of retrieval, causing the total problem solving time to increase.

In meta-level problem solvers, control rules improve performance by reducing the number of states that the problem solver visits, but this improvement comes at the cost of the additional overhead O_K of matching those control rules at *every state* that the problem solver visits:

$$t_K = (c+O_K)w_K$$

Therefore, in meta-level problem solvers, when the calculated speedup factor f_C of a knowledge base **K** is defined to be the ratio of the work of guided and unguided problem solving:

$$f_C = w_K/w_\varnothing \ ,$$

the actual speedup f_a can be defined as the ratio between the time cost of guided and unguided problem solving steps:

$$f_a = t_K/t_\varnothing \ = (c + O_K)w_K/cw_\varnothing \ = (c+O_K)fc.$$

Thus, the slowdown factor *S* can be computed as a function of the overhead O_K:

$$S = c+O_K.$$

Lest the reader think that we are stacking the deck to make our point, we note in passing that some systems explicitly assume an *ideal cognitive model*, in which the time of individual problem solving steps always remain constant (Tambe et.al. 1990).

Similar metrics can be defined for other types of systems, such as case-based reasoning systems. The retrieval of a case occurs only once in an ideal basic case-based reasoner, and so the total overhead is proportional only to the overhead of a single retrieval:

$$t_K = O_K + cw_K$$

The calculated and actual speedup can be again derived in the same fashion:

$$f_C = w_K/w_\varnothing \ ,$$

$$f_a = t_K/t_\varnothing \ = (O_K+cw_K)/cw_\varnothing \ = (c + O_K/w_K)fc.$$

Thus, for case based reasoners, the slowdown factor *S* is:

$$S = c + O_K / w_K.$$

As we can see, the overhead of retrieval creates much less of a burden on a case-based reasoning system than it does on a meta-level problem solver, because the cost of retrieval is *amortized* across all the remaining problem steps. Unfortunately, this advantage may disappear on more realistic, more complex case-based reasoners, especially those that have a high degree of recursion; furthermore, the cost of a retrieval of a case from a large case library may be greater of a single control rule.

4. A TAXONOMY OF UTILITY PROBLEMS

Given this formalism, we can identify two main varieties of the utility problem: performance utility problems, and search-space utility problems.

4.1. PERFORMANCE UTILITY PROBLEMS

We can define a performance utility problem as any utility problem in which the slowdown S is a function of the overhead O; that is, any utility problem that arises because of interactions between the learning that a system performs and the increased overhead the learning places on the system's architecture. While a knowledge base K may reduce the work that the problem solver does, the overhead it imposes outweighs that improved performance, increasing total problem solving time. There are two primary performance utility problems: the *swamping* problem and the *expensive chunks* problem.

4.1.1. SWAMPING

Swamping was first noticed in PRODIGY/EBL (MINTON 1988). 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 (COHEN 1990). In other words, the calculated speedup F provided by the database is outweighed by the slowdown $S(O_K)$ of the database. Each individual rule may be useful and easy to match, but the overhead of matching the rule set outweighs any performance benefit provided by the individual rules.

Swamping is, in a sense, the "core" utility problem because of the severe cost of retrieval in serial computers; the O(n) bound on matching alone can be crushing for large knowledge bases. For this same reason, it is the most amenable to solution by the addition of parallelism, and will be the primary focus of this paper.

4.1.2. THE EXPENSIVE CHUNKS PROBLEM

The *expensive chunks* problem was first noticed in the Soar system (TAMBE ET.AL. 1990). The problem occurs when single control rules are so expensive to match that they slow down the entire system. The expensive chunks problem occurs whenever a small subset of **K**, denoted **K'**, degrades system performance enough to outweigh the benefits of **K** as a whole. In other words, the calculated speedup F is again outweighed by the slowdown $S(O_K)$, but the primary contribution to that slowdown is the overhead $O_{K'}$ of a small *subset* of the rulebase **K'**, even though the size of **K'** may be very small ($|\mathbf{K}'| << |\mathbf{K}|$). In fact, the overhead of the rest of the knowledge base ($O_{K-K'}$) may not cause a slowdown at all (TAMBE ET.AL. 1990).

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.

4.2. SEARCH-SPACE UTILITY PROBLEMS

There are a number of other utility problems that can affect a meta-level problem solver that are unrelated to the performance overhead O_K. These utility problems arise because of problems in the way that the control rules modify the system's search through the search space, not because

of changes that occur in the system's architectural performance.

The *composability* problem occurs when individual control rules, learned to improve problem solving performance with regards to one metric, actually degrade problem solving performance as a rule *set* with regards to the *same* metric (GRATCH & DEJONG 1991A). The performance overhead is irrelevant here, because while the calculated speedup $F<1$ for each individual rule $k_i \in$ **K,** the total speedup of the whole problem base $F>1$ — that is, there is no speedup at all, calculated *or* actual.

The *branching* utility problem can arise when a problem solver learns macro-operators that change the topology of the search space itself. Etzioni effectively demonstrated that 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).

Finally, 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).

Solving these utility problems requires a careful analysis of the type of rules or cases that the system generates. For the composability problem in particular, this is extremely difficult, because control rules can theoretically interact in many different ways. A full treatment of search-space utility problems is beyond the scope of this paper, but, in general, search-space utility problems are best solved by never allowing them to occur in the first place (GRATCH AND DEJONG 1991B). Even if they are solved, however, this is no guarantee that the system will avoid performance utility problems. Therefore, the remainder of this paper will focus on parallel solutions to the expensive chunks and swamping problems.

5. A BRIEF REVIEW OF PARALLEL ARCHITECTURES

Both the swamping problem and the expensive chunks problem arise when the cost of the matching algorithm becomes so large that the speedup provided by the rules or cases is swamped by the overhead of matching them. One possible solution to the utility problem, then, would be to eliminate this overhead, perhaps through the use of massive parallelism to reduce the cost of matching to a constant-time operation in the number of rules. To determine whether on not this is possible, it is necessary to construct and analyze theoretical models of parallel machines. In this paper, we will examine a standard model of parallel systems called Parallel Random Access Machines, or *PRAMs*.

A *p*-processor PRAM consists of *p* ordinary serial processors, each of which can access a shared, global memory. Each processor can read from or write to any location in the global memory (FORTUNE AND WYLLIE 1978). For the sake of simplicity, all processors are assumed to be synchronized with each other, so that every individual processor performs exactly one serial operation within the same time slice.

It is often convenient to assume that each processor p_i is "responsible" for a corresponding memory location, denoted m_i. In practice, there may be far more memory locations than processors; the additional space is used for "scratch" computation. For this text, we will assume that each processor p_i is responsible for two memory locations: a storage location k_i containing an item in the knowledge base **K**, and a scratch location m_i used to store intermediate results in computations.

The basic PRAM model does not define what happens if two processors attempt to read from

or write to the same memory location at the same time. Both reading and writing can be either exclusive or concurrent operations, and there are four flavors of PRAMs based on all possible combinations of exclusive and concurrent reads and writes. For the purposes of this paper, we need only concern ourselves with the two extremes: *exclusive read, exclusive write* (EREW) systems, and *concurrent read, concurrent write* (CRCW) systems. In an EREW system, only one processor can read from or write to a given memory location at any one time; in contrast, any number of processors can read from or write to a given memory location at any one time in a CRCW PRAM (CORMEN ET.AL. 1990).

Implementing a large, ideal PRAM of either variety is very difficult, and many theoreticians believe that large, completely connected parallel machines are impractical (PARBERRY 1987). Faced with these limitations, researchers have considered more tractable models of parallel machines that require less connectivity, including models like the *hypercube* (SHIH AND FIER 1989). Hypercubes are multiprocessor parallel computer with a *distributed*, rather than global memory; a hypercube of dimension d has 2^d processor nodes, each connected to d other processor nodes. Even extremely large hypercubes can be constructed well within the limits of technologies under development — a hypercube with 10^9 processors, for example, would only require on the order of 10^{10} interconnections, well within the upper limits currently projected for optical computing (HECHT-NIELSEN 1986). A PRAM of 10^9 processors, on the other hand, would require 10^{18} interconnections.

Finally, it is important to note the blurry distinction between serial and parallel systems. Parallelism is only relative to the size of the knowledge base; a parallel machine with 10 processors does not provide many of the traditional advantages of parallelism if the expected size of the knowledge base ranges from 1,000 to 10,000 items. On the other hand, a parallel system that can guarantee that no processor will ever have to handle more than 10 items may provide many of the same benefits as a system that allocates one processor per data item. For the purposes of this paper, we define a parallel machine to be a PRAM whose number of processors is on the order of the number of items in the knowledge base (p is O(n), where n is the size of the knowledge base), and a serial machine to be a PRAM whose number of processors is constant (p is O(1) in n).

6. PARALLEL CASE RETRIEVAL

The two core operations of retrieval can both be aided by a move to parallel hardware, but there are pitfalls with each operation that complicate the benefits of parallelism.

6.1. PARALLEL MATCHING OF CASES AGAINST AN INDEX

PARALLEL-MATCH
Given a knowledge base K = ($k_0..k_{n-1}$), a state x, a matching function g, and a match threshold t:
1) **For all** $i = 0$ **to** $n-1$ **in parallel do** $m_i = g(x,k_i)$

On serial machines, Parallel-Match is O(k) with respect to the size of the casebase, n. On any PRAM, Parallel-Match can be performed in O(1) time with respect to the size of the casebase, n. However, this glosses over a number of other possible contributions to the running time. For this discussion, the most important contribution comes from the time needed to perform the matching function g on each individual case. If the size of each case is unbounded, then the time it takes to match each case is similarly unbounded (ETZIONI 1992; TAMBE ET.AL. 1990). This is the expensive chunks problem; while matching all the cases can be done in parallel, individual cases can cause the total time for matching to increase.

Parallelism does provide a vast improvement over serial systems, in which match time is the

sum of all the match times of all the individual cases. On a parallel system, the time of matching is simply the maximum match time of any individual case; however, this only a partial solution to the expensive chunks utility problem; because the total match time of an individual case is still unbounded. These results can be summed up by the following observation:

Observation 1: If rule size is unbounded, the running time of Parallel-Match, while still O(1) in n, can become arbitrarily large.

6.2. PARALLEL SELECTION OF CASES FOR ADAPTATION

There are a number of different algorithms for selecting a memory item out of matching items in a database; for case-based reasoning systems, however, the most appropriate policy is *Best-Selection* — selecting the case that matches the current state best. Meta-level problem-solvers can use different retrieval policies, such as *Arbitrary-Selection* — selecting any rule that matches within some tolerance value — and *Absolute-Selection* — select all rules that match within some tolerance value.

Implementation of Arbitrary-Selection selection can be problematic — for example, Arbitrary-Selection may select a preference rule for a given operator over a higher-matching rejection rule for the same operator — and implementing Absolute-Selection can lead to the problem of a large number of rules concurrently executing at once. Because Arbitrary-Selection is unsuitable for most case-based reasoning systems and Absolute-Selection cannot be implemented at all, we will restrict our attention to the Best-Selection algorithm.

BEST-SELECTION(A)
Given knowledge base **K** = ($k_0..k_{n-1}$), a state *s*, and a matching function *g*:
1) **For all** i = 0 to n-1 **in parallel do** $m_i = g(x,k_i)$
2) **For all** i = 0 **to** n-1 **in parallel do** select the rule with the best match score for execution

---

Step 1 of BEST-SELECTION is simply PARALLEL-MATCH, so the same complexity bounds and the same observations hold: Step 1 of BEST-MATCH is O(1), but the constant factor can become arbitrarily large if rule size is unrestricted.

Step 2 is more problematic. Finding a single best match is essentially the same as finding the maximum of a set of values. If concurrent writes are allowed, finding the maximum can be done in O(1) time; for exclusive-write PRAM's, however, the lower bound on finding the maximum element of an array is $\Omega(\lg n)$, even with unlimited processors and unlimited memory (COOK ET AL 1986).

Alternative models of parallelism fare no better than exclusive-write PRAM's; the best that a hypercube can do on either BEST-SELECTION is O(lg n) time. While actually performing the matching operations can be done in constant time in a hypercube with one processor per data item, propagating the match criteria to every node would take $\Omega(\lg n)$ time. Selecting the best match is also a $\Omega(\lg n)$ time operation. This is no better than an exclusive-write PRAM, and possibly much worse.

Observation 2: On any parallel machine prohibiting concurrent writes, BEST-SELECTION is $\Omega(\lg n)$.

7. CONSTRAINTS ON HARDWARE SOLUTIONS TO THE UTILITY PROBLEM

We began our analysis of parallel systems by categorizing them in terms of the numbers of processors and interconnections that the systems had and the types of operations that the system permitted. Now, we can devise a new categorization, based not on the internal architecture of the system but instead on the running times for retrieval that the architecture permits.

The worst class of systems are *Class I* retrieval engines, in which retrieval time is O(n) in the

knowledge base size. These are current serial systems and most current parallel systems. *Class II* systems improve upon this by reducing retrieval time to O(lg *n*); hypercubes and EREW PRAMS meet this criterion. *Class III* retrieval engines, the most powerful class of systems, have a retrieval time that is constant and independent of the size of the knowledge base.

	Class I Systems	Class II Systems	Class III Systems
Hardware Types	Serial RAMs Small PRAMs,	EREW PRAMs, Hypercubes	CRCW PRAMs
Parallel-Match Time	O(n)	O(1)	O(1)
Best-Selection Time	O(n)	O(lg n)	O(1)
Total Retrieval Time	O(n)	O(lg n)	O(1)

Table 1. Running times of algorithms for various types of hardware

Clearly, Class III machines have the potential to fulfill the expectations of an ideal cognitive model proposed in some systems, such as ABBY (DOMESHEK 1992) and Soar (TAMBE ET.AL. 1992). However, there are severe constraints on the kinds of hardware that can live up to the stringent requirements of the CRCW PRAM model needed to implement a Class III machine.

In current (VLSI) systems, it is unclear whether or not CRCW machines really provide the advantages that they theoretically claim to, since CRCW hardware is, in general, slower than EREW hardware. The full power of a concurrent-write PRAM depends essentially on its ability to perform a Boolean AND of *n* values in constant time (CORMEN ET.AL. 1990). In VLSI technology, however, the lower bound on the size of a Boolean formula in *n* variables is $\Omega(n)$, excluding fan-out considerations of individual processing elements (BRENT AND GOLDSCHLAGER 1982). Including fan-out considerations upper bound on the area of such a chip is $O(n^2 \log^3 n)$. For time, the upper bound is the familiar O(log *n*). These considerations indicate that, at least, the real silicon hardware to implement the interconnections of a concurrent-write PRAM will be more expensive than the processors used in computation. At worst, it indicates that the performance advantages of VLSI concurrent-write PRAMs are entirely imaginary, because the hardware required to run them eliminates the advantages of the abstraction.

Optical computing may provide a way of getting around the AND problem; optical AND gates of virtually unbounded fan-in can be constructed simply by directing multiple beams of light towards a single source (TOBORG AND HWANG 1989). However, despite their projected speed and versatility, even optical computers fail to provide the number of interconnections needed to support a full PRAM of billions or even millions of elements. Volume holograms are projected to provide up to billions of interconnections (HECHT-NIELSEN 1986) but even a million-element PRAM would require a trillion interconnections.

Beyond optical computing lie more exotic technologies. Molecular nanotechnology will make possible the inexpensive mechanosynthesis of ~1000 MIPS nanomechanical RISC systems in extremely short time scales (DREXLER 1992). This will make the construction of parallel systems with extremely large numbers of processors easier, perhaps even allowing systems to be extended on the fly. Furthermore, if some analog method of computing an AND with near-unbounded fan-in can be devised, constructing the requisite number of interconnections can be performed cheaply and reliably.

The point of this section is not that it is impossible to build a giant concurrent-write PRAM; similar remarks were made about airplanes around the turn of this century *after* the Wright Brothers' first flight. The point is that designing a large Class III machine is a highly nontrivial problem, requiring 1) massive parallelism, on the order of the number of items in the database and either 2a) full connectivity, or 2b) dedicated hardware for the distribution of match criteria to all

nodes (a dedicated equivalent of a concurrent read) and dedicated hardware for the accumulation of best-match rules, (a dedicated equivalent of a concurrent write).

This is possible, but far in the future. Even if a Class III machine were built, it would be an extremely expensive solution, as measured in the sheer amount of processor power and other hardware devoted just to *interconnections* rather than actual *computations*. For this reason, it is likely that Class II machines — which only require the number of processors, not the full connectivity — will be built and become common before any Class III machines are ever constructed. The practical effect of this can be summed up by the following observation:

Observation 3: Theoretical and practical considerations currently do not permit the construction of Class III retrieval engines, placing a lower bound of at least O(lg n) on retrieval algorithms on parallel hardware.

8. The Significance of the Time Complexity of Best-Selection

A logarithmic time bound is usually a joy to the eyes of any computer scientist; it usually signifies an easy problem, one readily amenable to increased processor size and time. For machine learning, however, the logarithmic time bound means that the utility problem will not go away, at least for multi-purpose learners functioning in an open world. The problem arises and remains because the absolute speed of the memory is not the issue — only the relative speedup is.

In speedup learning, the most that a learned item can provide is a speedup of a constant factor *F*. To see this, note that while a casebase or rule set may reduce exponential search to polynomial-time problem solving, for *any one given problem instance* the problem difficulty is fixed, and the ratio of unguided and guided work performed is similarly fixed. This ratio, *F*, depends only on the problem, the problem difficulty, and the advantages provided by the rules that apply to the particular problem.

8.1. Problem Solving in Closed Worlds

Closed world systems assume that a reasoner's knowledge about a domain in which it operates is complete — one in which input features are set in advance, input data is quiet (error-free), and the domain knowledge needed for problem solving can be completely specified in advance (Rieter 1978). In a closed world system with limited-size problems, a problem solver can always converge into a finite set of rules to solve a problem.

To see why, let the input size of the problem be its length in bits. If problems are less than some size *s*, then there are less than 2^s problems. A rote case-based reasoning system could store each case identically, recalling it if and only if the same problem statement appears. Of course, such a rote learner is intractable. Suppose the maximum problem size is 100 bits, not an unreasonable assumption. Then we have 2^{100} problems, which is $(2^{10})^{10} \cong 10^{30}$ different problems.

On a serial system, the total overhead for such a knowledge base is $O(2^s)$, which is an exponential in problem size and likely to quickly exceed the work of rederiving a solution. On a Class II parallel system, however, the total overhead would be $O(\lg 2^s) = O(s)$, which is polynomial in problem size. This is a large improvement, but problem size has lttle to do with problem difficulty. If the rote learner's problems are not that difficult, then the rote learner still suffers from the utility problem. For example, assume that the cost of applying an operator is roughly equivalent to a memory retrieval of a single case, and further assume that the result of each problem was a 1-step path, obtained from applying the result of one of ten operators. Then, the time cost of unguided problem solving is simply 10. Then memory will experience a slowdown of a factor of 100 over the course of learning, and will very quickly exceed the cost of applying

and testing the results of each operator.

However, assuming that the problem statement *is* tractable and assuming that the problems have a large difficulty values, then we can construct a rote learner or other simple learning system running on Class II parallel hardware that will learn all that it needs to know without ever encountering the utility problem.

8.2. Problem Solving in Open Worlds

In open worlds, many of the safe assumptions of closed worlds do not apply because a reasoner's knowledge may be incomplete or inconsistent (Hinrichs 1991). For example, the set of input features may need to be learned; alternatively, a set of input features may be specified in advance, but the number of bits needed to describe a problem can be incredibly large, even into millions of bits (Domeshek 1992), in which case a rote learner is intractable. Input may be errorful, or the existing set of features may not cover all of the relevant features of the domain. The domain theory may not be complete or may not exist.

In an open world, it is unlikely that a learner would ever converge on a set of learned rules or cases that completely cover the domain. Even if such a set exists, the system's input bandwidth and useful life may not be long enough to acquire it; the universe is an example of such a domain with respect to human science.

Even if such a learning system were able to acquire a complete set of rules for some subtask in the problem domain, it will continue to learn about other problems and other intricacies of the system. In effect, the learner runs forever, and the maximum problem size (the maximum amount of data you can apply in any given situation) may be very small compared to the number of examples the system encounters in its lifetime.

In open worlds, the slowdown S caused by the overhead of the performance component is a function of the size of the database and can increase independently from F as the system learns about other problems in other domains. While speedup remains fixed, the slowdown increases at a rate of at least lg n and is unbounded in size. This leads us to the following observation:

> *Observation 4: Any policy for learning items only guaranteed to bring an average speedup of F will be swamped in the limit by the increasing cost of memory access for any parallel machine that prohibits concurrent writes.*

A valid criticism of this characterization of the utility problem is that no current algorithms ever approach the number of items in their databases that they would need to cause swamping on a parallel machine. This is only partially true; no existing system that suffers from the utility problem has ever been scaled up that far, but databases that would cause the problem already exist — for example, the CYC database (Lenat and Guha 1992). Even those who do not agree with the CYC philosophy must recognize that the *size* of the CYC database is a good indication of how large AI databases are likely to get *on open world problems*. Case bases are likely to grow to this size as well, especially in general-purpose case based reasoning systems that depend on extensive databases in multiple domains to provide cross-domain remindings. This leads to the next observation:

> *Observation 5: Existing learning algorithms produce learn items with an average speedup that is small enough that even a logarithmic slowdown will swamp them for reasonably-sized databases and open world problems.*

For example, the CYC database currently holds 10^7 items, and even that number reflects an incomplete database not yet containing any additional modules for application domains. It is a reasonable extrapolation to assume that the CYC database could grow in size to as many as 10^9 items, especially if the underlying model is extended to include speedup learning. 10^9 items will cause a $\Omega(\lg n)$ slowdown on the order of ~30, regardless of the constant factors, which cancel

out when computing slowdown. But the average speedup provided by various learning schemes is far less than this — on the order of 6-10 times for Soar, and 2-6 for PRODIGY. This indicates that if Soar or PRODIGY are scaled up to reasonably-sized open world databases, any rule they learn is *guaranteed* to cause slowdown, rather than speedup.

8.3. What About the Constant Factors?

One part of the picture hidden by the O(lg *n*) time bound is the constants of memory retrieval and problem-solving steps. If each problem solving step takes 100 times as long as a memory access, then even a slowdown of 30 on memory operations will not cause an overall slowdown in average problem solving time. On the other hand, if memory operations take as long as or longer than problem solving steps, then even a small slowdown can cause the utility problem.

For conventional hardware, memory accesses are generally slower than CPU operations. This problem tends to worsen as CPU speeds increase because memory access generally require off-chip access. Even if we assume that memory operations can be brought on-chip, however, it is reasonable to assume that memory accesses may still require more time. As interconnections increase, the size and time required to input values increases at $O(n^2 \log^3 n)$ and $O(\log n)$ respectively. For a fully connected PRAM, each interconnect may have a billion input bits, and for most problems the space and time complexity of such a circuit will be far larger than the space and time complexity of the (relatively) simple circuitry needed to perform an individual problem step.

9. Coping Strategies and Case Based Reasoning

Given the nature of the utility problem and the theoretical limits and practical limits on parallel solutions to the problem, it is clear that alternative solutions are needed. However, the pitfalls of the idealized matching algorithms considered earlier — the arbitrarily large constant factor that arises during the matching of rules and the O(lg *n*) bound for finding the best rule in the entire database — provide us with fairly good indications of the kinds of solutions that will be effective in eliminating the utility problem.

Because the ideal algorithms are intractable, solutions to the utility problem lie in deliberately adopting explicit storage and retrieval policies that depart from the ideal algorithms in ways that are guaranteed to improve performance. The following list of policies is by no means exhaustive, but they do suggest ways to attack the utility problem.

9.1. Swamping

1) Class III Retrieval Engines: Use a hardware solution to provide an ideal cognitive model. Tempting though this policy is, it is not currently available and will not be within the lifetimes of current AI systems.

2) **Restricted Learning Policy**: Never learn anything with a speedup of less than the slowdown produced by the maximum expected size of the system database. In other words, compute an upper bound on the overhead *O* incurred by the memory and the corresponding slowdown *S* that this causes; then, ensure that the speedup factor *F* will always be greater than the maximum slowdown. For systems with quick matching times and long problem-solving times this may be relatively easy. For memory-intensive systems, this may be nearly impossible.

Case-based reasoning provides two primary kinds of support for a restricted learning policy. First, by amortizing the overhead over many adaptation steps, ideal case-based reasoners suffer less severely from the same overhead than a more conventional meta-level problem solver. Second, cases have the potential to eliminate vast amounts of search in the search space, providing improvements robust enough to survive the architectural slowdown.

An example of a system that satisfies these requirements is the CASEY program (KOTON 1989), a case-based reasoning system that serves as a speedup module for a model-based medical expert system. CASEY's cases provided up to three orders of magnitude of improvement over the original problem solver. Even a slowdown of a factor of 30 or more would not swamp the speedup provided by such a case, assuming that case memory access time is within the same order of magnitude as the time needed to adapt a case. Unfortunately, as noted above, most problem solving systems do not provide performance improvements anywhere near this order of magnitude.

3) **Deletion Policy**: Prune the database so that the slowdown never becomes unreasonable. Formally, fix a lower bound on *F*, the minimum acceptable speedup, and then constrain the size and overhead of the database such that *S* never exceeds *F*. Deciding what can and cannot be deleted is in and of itself a difficult problem. For some systems, like Prodigy, in which no one operator or rule is critical, deletion of least-recently-used or even random items may be a viable option. Experiments with the FUNES system verify that even random deletion can improve performance (MARKOVITCH AND SCOTT 1988). Other systems, like Soar, combine the results of knowledge-level and symbol-level learning in the same rule format; the elimination of the wrong rule can cause critical errors in a problem-solving sequence.

Case-based reasoning systems can adapt to a deletion policy, but most real case-based reasoning systems contain critical knowledge in many of their cases, knowledge without which solutions are impossible. Deletion policies in these kinds of systems must be carefully designed never to eliminate knowledge that is critical to the system's operation.

4) **Guided Search Policy:** Limit the amount of the database that is examined during retrieval so that the slowdown never becomes unreasonable. Formally, this is similar to a deletion policy, except that it is not the size of the database itself that is being constrained but instead the *amount* of the database that is searched. Matching and/or selection is constrained to a subset of the database; for the purposes of the running time of retrieval, it is as if the system was operating with a smaller database and incurring a proportionately smaller overhead.

However, no information is ever eliminated from the database; if critical information that is needed is not found on the first pass of retrieval, then retrieval can be extended to find it without imposing a performance penalty on all retrievals that find a suitable item on the first pass. Various policies of guided search have proven effective in speeding up problem-solving performance (MOONEY AND SRIDHARAN 1989, MARKOVITCH AND SCOTT 1989).

The real advantage of guided search policies is that they scale smoothly to both Class I and Class II machines (JONES 1992). (Scaling to Class III machines is not necessary, of course, because the problem goes away.) Search needs to be more severely curtailed for serial than parallel implementations to gain the same benefit, but the processes involved are identical.

A number of case-based reasoning systems already implement guided search policies through their choice of indexing schemes (KOLODNER 1992). These indexing schemes partition the system's case library and use indices — processed match criteria — that allow the system to search only a subset of the available partitions. Indexing thus makes case-based reasoning resistant to architectural overhead based purely on the increase in numbers of cases, and makes case-based reasoning a viable learning strategy for both serial and parallel machines.

9.2. EXPENSIVE CHUNKS:

1) **Restricted Expressiveness Policy:** Eliminate expensive chunks by limiting the kinds of chunks that the system can form. In Soar, this was accomplished by limiting the language used to express chunks so that expensive chunks were never formed; this tack was successful for a wide range of problems (TAMBE ET.AL.1990). One of the drawbacks to this policy is that more chunks

need to be formed to provide the same speedup; however, experiments with Soar indicate that this is an effective means for eliminating the utility problem.

Case-based reasoning systems can suffer from the expensive chunks problem, but systems can easily incorporate a restricted expressiveness policy into the indexing scheme itself. For example, some case-based reasoning systems attack the expensive chunks problem by placing an upper bound on the size of an item that can be matched (DOMESHEK 1992).

2) **No-Wait Policy:** Terminate longer-running matches when a successful alternative has been found. This is really a combination of Absolute- and Best-Selection policies. If a quick-matching rule has a high match score, relative to some absolute scale, it is posted to the problem solver and matching of all other rules is terminated. Case based reasoning systems can easily adapt to a no-wait policy.

3) **Asynchronous Match Policy:** Allow expensive chunks to match asynchronously. One way to do this and still preserve the "best match" philosophy is to have a cut-off point at which all the completed matches are compared against each other. The maximum value is posted to the problem solver, but if any later matches complete with a better match value, they are posted to the problem solver as well as they come in, like news-flashes in a newsroom. Also like a newsroom, there may be an absolute deadline for a match time — for example, the point at which a matched rule's preconditions no longer apply.

Most case-based reasoning systems cannot take advantage of asynchronous match policies, because they only select a single case for adaptation at once. Case-based reasoners that attempt to apply and adapt multiple cases could, however, allow matching and retrieval to continue in the background while the intial case is being matched, and then would conceivably have a better set of cases to choose from when it selected the next case for adaptation.

9.3. COPING STRATEGIES FOR OTHER UTILITY PROBLEMS

In addition to their advantages on performance utility problems, case-based reasoning systems have a number of other features that aid them in tackling search-space utility problems as well. For example, basic case-based reasoners avoid the composability problem entirely, because two cases cannot be applied at once. More advanced case-based reasoners may call case-based reaosning recursively and might run into a situation where two adaptation cases may conflict; once a conflict is recognized and resolved, however, it could be stored in the case itself to eliminate conflicts in the future.

Furthermore, basic case-based reasoning systems avoid the branching problem by segregating learned cases from operators. Applying an operator and selecting a case are entirely different operations. Learning a new case may increase the overhead of the memory system, but it does not increase the branching factor of the search space, because cases are not operators and cannot be applied in the place of operators. Even more advanced case based reasoners that learn macro-operators for adaptation can also store preference and selection rules for operators with each case, so that the application of these new operators is restricted and the increase in the branching factor eliminated.

More advanced case-based reasoners can avoid the wandering utility problem by learning from both success *and* failure. Advanced case-based reasoning systems can assign blame in failure scenarios and learn critics that use this knowledge to terminate adaptation paths likely to end in failure; similarly, analysis of successes can lead to preference critics that select more likely paths of adaptation first, increasing the likelihood that a system will more quickly reach a solution.

10. FURTHER WORK

10.1. A NEW METRIC: CRITICALITY OF KNOWLEDGE

Not all learning is speedup learning; some systems combine knowledge-level learning and symbol-level learning, sometimes even using the same knowledge representation format for both. In this case, both deletion and guided search policies must be devised carefully, because the overhead swamping out the utility of control rules may actually come from a body of domain knowledge which may be critical for the system's operation. In these systems, knowledge must be characterized on two dimensions: its *utility* and its *criticality*.

The utility of a piece of knowledge is typically defined as the degree to which the knowledge alters the problem solver's search through the problem space; in contrast, the criticality of a piece of knowledge can be defined as the degree to which it *alters* the problem space itself. Thus, *critical* knowledge is knowledge that *defines* the problem space within which a system functions. *Semi-critical* knowledge, like cases and macro-operators, redefines the *topology* of the search space, but does not redefine the set of *states* that the search space contains. *Non-critical* knowledge does not redefine the problem space, but it may alter the search of that space significantly.

In a true meta-level problem solver, critical and non-critical knowledge are segregated: all critical knowledge is always contained within the problem solving component, and all non-critical knowledge is contained in the knowledge base itself. In traditional case-based reasoning programs, the case library usually contains both non-critical and semi-critical knowledge. If the best-match case is deleted, these systems can usually adapt the next-best match, possibly with reduced performance or competence. In some systems, such as MEDIATOR, every case may contain critical knowledge, without which entire portions of the search space are inaccessible (KOLODNER 1989).

A thorough analysis of the general utility problem must analyze the relationship of criticality of knowledge to various policies for restricting search and matching. One way to do this might be to use the system's learning goals to identify and organize useful knowledge for selective storage and retrieval (RAM AND HUNTER 1992). A formal characterization of criticality is crucial for devising effective guided search and deletion policies.

10.2. MOORE

As an attempt to put the principles embodied in this paper into practice, the authors are currently constructing a memory module called MOORE which attempts to eliminate the utility problem. MOORE uses a combination of restricted expressiveness and asynchronous match policies to limit the expensive chunks problem, and adopts a guided search policy called *context focusing* to limit swamping. Context focusing uses a trace of a system's reasoning both within and across problems to guide memory search. This trace, along with a record of the effectiveness of memory items for retrieval, is also used to index new memory items as they are stored. A potential advantage of context focusing is that it is both an indexing scheme *and* a guided search policy; in theory, context focusing can scale smoothly to all three classes of hardware, from serial to exclusive-write to concurrent-write systems, without the need for reprogramming or reprocessing of the underlying database.

ACKNOWLEDGMENTS

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

REFERENCES

(BAASE 1988) Baase, Sara. *Computer Algorithms: Introduction to Design and Analysis, 2e.* Addison-Wesley Publishing Company, 1988.

(BRENT AND GOLDSCHLAGER 1982) Brent, Richard P. and Goldschlager, Leslie M. "Some Area-Time Tradeoffs for VLSI." *SIAM Journal on Computing*, 11(4):737-747, 1982

(COHEN 1990) Cohen, W.W. "Learning approximate control rules of high utility." In *Machine Learning: Proceedings of the Seventh International Conference*, 1990.

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

(CORMEN ET.AL. 1990) Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. *Introduction to Algorithms*. The MIT Press, McGraw-Hill Book Company. 1990.

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

(DREXLER 1992). Drexler, K Eric. *Nanosystems: Molecular Machinery, Manufacturing, and Computation*. John Wiley & Sons, 1992.

(ETZIONI 1992) Etzioni, Oren. "An Asymptotic Analysis of Speedup Learning." In *Machine Learning: Proceedings of the Ninth International Workshop*, 1992.

(FAHLMAN 1988) Fahlman, Scott E. "Parallel Processing and Artificial Intelligence." In Kowalik, ed. *Parallel Computation and Computers for Artificial Intelligence.* Kluwer Academic Publishers, 1988.

(FORTUNE AND WYLLIE 1978) Fortune, Stephen and Wyllie, James. "Parallelism in random access machines." In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, p114-118, 1978.

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

(GRATCH AND 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, USA, August 1991.

(HAMMOND 1989) Hammond, Kristian. *Case-based Planning: Viewing Planning as a Memory Task*. Academic Press, New York, 1989.

(HECHT-NIELSEN 1986) Hecht-Nielsen, R. "Performance Limits of Optical, Electro-Optical and Electronic Neurocomputers." Optical and Hybrid Computing, SPIE, v634, p277-306, 1986.

(HINRICHS 1991) Hinrichs, Thomas Ryland. "Problem Solving in Open Worlds: A Case Study in Design." *Thesis GIT-CC-91/36*. Georgia Institute of Technology, September 1991.

(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.

(JONES 1992) Jones, Randolph. "Problem Solving via Analogical Retrieval and Analogical Search Control." In Chipman and Meyrowitz, eds., *Machine Learning: Induction, Analogy and Discovery*. 1992.

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

(KOLODNER 1988) Kolodner, Janet Lynne. "Retrieving events from a case memory: A parallel implementation." In Kolodner, Janet Lynne, ed., *Proceedings of the 1988 DARPA Workshop on Case-Based Reasoning*, p233-249, 1988.

(KOLODNER 1989) Kolodner, Janet Lynne. "The MEDIATOR: Analysis of an Early Case-Based

Problem Solver." *Cognitive Science*, 13, p507-549, 1989.

(KOLODNER 1992) Kolodner, Janet Lynne. *Case Based Reasoning.* ?Publisher?, 1992.

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

(MARKOVITCH AND SCOTT 1988) Markovitch, Shaul and Scott, Paul D. "The Role of Forgetting in Learning." In *Machine Learning: Proceedings of the Fifth International Conference*, 1988.

(MARKOVITCH AND SCOTT 1989) Markovitch, Shaul and Scott, Paul D. "Utilization Filtering: a method for reducing the inherent harmfulness of deductively learned knowledge." In *IJCAI-89: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

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

(MOONEY AND SRIDHARAN 1989) Mooney, R. and Sridharan, N.S. "The effect of rule use on the utility of explanation-based learning." In *IJCAI-89: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

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

(RAM AND HUNTER 1992) Ram, Ashwin and Hunter, Lawrence. "The use of explicit goals for knowledge to guide inference and learning," *Applied Intelligence*, 2(1), 1992.

(RIETER 1978) Rieter, R. "On Closed-World Databases." In FGallaire, H, and Minker, J., editors, *Logic and Databases*, p55-76. Plenum Press, New York, 1978.

(SHIH AND FIER 1989) Shih, Yin and Fier, Jeff. "Hypercube Systems and Key Applications." In Kai Hwang and Doug DeGroot. *Parallel Processing for Supercomputers and Artificial Intelligence.* McGraw-Hill, 1989.

(TAMBE ET.AL. 1990) Tambe, Milind; Newell, Allen; Rosenbloom, Paul S. "The Problem of Expensive Chunks and its Solution by Restricting Expressiveness." *Machine Learning*, v5 p 299-348, 1990.

(TOBORG AND HWANG 1989) Toborg, Scott T. and Hwang, Kai. "Exploring Neural Network and Optical Computing Technologies." In Kai Hwang and Doug DeGroot. *Parallel Processing for Supercomputers and Artificial Intelligence.* McGraw-Hill, 1989.

(VELOSO & CARBONELL 1993) Veloso, Carbonell. "Paper." *Machine Learning*, v10 p?-?, 1993.