Systematic Experimentation with Deductive Learning: Satisficing vs. Optimizing search

Shaul Markovitch and **Irit Rosdeutscher**
Computer Science Department
Technion - Israel Institute of Technology
Haifa, Israel

Email: shaulm@cs.technion.ac.il
iritr@cs.technion.ac.il

Abstract

Most of the research conducted in the area of deductive learning is experimental. However, many of the experiments reported are far from being systematic and thorough. There are many parameters that are embedded in the system's architecture and it is not clear how they effect the utility of the learned knowledge. In this paper we describe an attempt to perform systematic experiments in the domain of deductive learning. The part described here explores how the search strategy employed during problem solving and during learning effects the utility of learning process. It was concluded that the utility of deductive learning is negative when the learned knowledge is applied in optimizing search procedure during problem solving, but becomes positive in satisficing search. It was also concluded that for off-line learning it is more beneficial to use optimizing search during the learning process. Knowledge acquired in this method improves both the efficiency and the quality of the problem solving.

1. INTRODUCTION

The main research methodology employed in the field of Machine Learning is experimental. Kibler & Langley (1988) give a thorough account of the importance of experimentation in the field. One of their conclusions is that the field of Machine Learning lacks systematic experimentation in the domain of problem solving. The work described in this paper tries to remedy this situation. We have developed a systematic set of experiments performed in the domain of deductive learning in order to better understand the nature of such learning processes.

A deductive learner is a process in which a system generates knowledge that can be deduced by its inference mechanism and stores the knowledge for later use. A large portion of existing learning systems that try to improve the efficiency of a problem solver use deductive learning (Iba, 1989; Markovitch & Scott, 1988; Markovitch & Scott, 1989; Minton, 1988; Mooney, 1989). A very common scheme used in such systems is to solve training problems during learning and acquire deductive knowledge during the search.

Most of the research done in this area depends on experimental results. The problem is that†few thorough experiments have been performed in the area of deductive learning (Kibler & Langley, 1988). In a large portion of the experimental work, many parameters†are embedded in the system's architecture and it is often unclear what the effects of the particular chosen values on the system's behavior are.

There are several important factors that may effect the usefulness of deductive learning:

- The nature of the algorithm used for search during problem solving.
- The nature of the algorithm used for search during learning.
- The nature of the space that the problem solver searches.
- The nature of the problems that the problem solver solves.
- The nature of the algorithm used for acquiring the deductive knowledge.

- The nature of the filters used.

We have developed a research program whose goal is to understand how each of these factors effect deductive learning, and what are the meaningful relationships between them. In order to do so we parametrized these factors, and then performed experiments varying various parameters while holding the rest.

The goal of this paper is to demonstrate how such parametric study should be conducted by describing experiments with two parameters: the search strategy used during learning and the one used during problem solving. We have tested how an optimizing search during learning effects the utility of the learned knowledge compared to a satisficing search (Simon & Kadane, 1975). The same questions were asked with regard to the search during problem solving.

To make the experiments feasible, and to make the results general enough to be applicable to all deductive learners, we had to make the following simplifying assumption:

> *The non-decomposability assumption*: The states in the search graph are atomic, i.e., a state cannot be decomposed into finer parts. We also assume that we do not have at our disposal any semantic information about states.

This assumption imposes significant restrictions on various aspects of the learning process and thus reduces the complexity of the experimentation. The most important restriction is that the learned knowledge cannot be generalized: there are no features to generalize. In addition, we have chosen a relatively simple search space and a simple macro learning procedure. Even with these simplifications, performing a parametric study is a very laborious research task.

Section 2 of the paper describes the components that were used for the experiments. Section 3 describes the parametrization of the problem. Section 4 describes the experiments performed. Section 5 concludes.

2. THE SEARCH SPACE, PROBLEM SOLVER AND LEARNING PROGRAM

2.1. THE SEARCH SPACE

Kibler and Langley (1988) noted that artificial domains have a major advantage over natural domains: they allow control of domain characteristics as independent variables. We were looking for a domain whose complexity could be controlled. This requirement excluded all known domains such as the 8-puzzle. We finally decided to experiment on a grid domain where each state is a junction in the grid and has at most 4 neighbors. We assumed a uniform cost of 1 for all edges in the search graph. For the experiments described in this paper we have used a grid of 50×50 that contains 2,500 states. We wrote a procedure that generates random parametrized grids by inserting walls into the grid. A wall is a section of a column in the grid that is disconnected: if there is a wall between two intersections that are neighbors, then there is no direct transition between the two.

All the walls are parallel - there are no walls along rows. This was done to make sure that the space will stay fully connected so that every search problem will have a solution. In this method we avoid the problem identified by Segre (1991) where insolvable problems corrupt the performance measurement.

The procedure that creates random spaces gets the following parameters for controlling the complexity of the space:

1. Wall-frequency : the probability that a column will be selected for wall building.
2. Wall-portion : the portion of a column that will be filled by walls.
3. Fraction size: the size of each fraction of a wall.

The parameters are controlled by lower and upper bounds, and are generated by selecting random values between the bounds. Figure 1 illustrates a typical grid space.

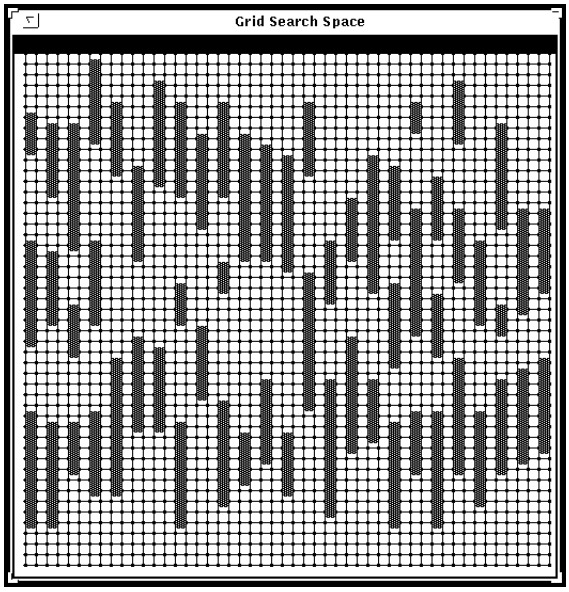

Figure 1: An example of a simple grid domain

We are aware that the search space is relatively simple: it contains only thousands of states while other search spaces such as 15-puzzle contains billions of states. However, at this stage of research our main concern is with being able to control and manipulate the space easily. Even the relatively small domain described above contains several millions of potential macros to learn.

The search graph described above is an OR graph. An important enhancement of the research would be to deal with AND/OR graphs. Such an extension will increase the complexity of the experimentation significatly.

2.2. THE PROBLEM SPACE

A problem is an ordered pair of states: the initial state and the goal state. For simplicity we do not experiment with problems that have multiple goal states. For a state space that contains N distinct states, there are N^2 possible problems, 6,250,000 for the grids used during the experiments. It would be very difficult to learn knowledge that is relevant to such a large set of possible problems.

To make the learning more focused, we have restricted the problem space to a much smaller set. This was done by defining two square areas on the grid: one 10X10 area in the middle of the left side of the grid, and one 10X10 area in the middle of the right side of the grid. Initial states are selected from the left area and final states are selected from the right area. Thus the size of the problem space is 10,000.

2.3. THE PROBLEM SOLVER

The search procedure that was selected for experiments is weighted-A*. The behavior of A* and its variants are reasonably well understood (see for example (Pearl, 1984)). The heuristic function used for all the experiments described here is the Manhattan-distance heuristic (Pearl, 1984)[1].

2.4. THE LEARNING PROCEDURE

When designing the learning procedure we tried to restrict ourselves to the most general algorithm that captures the essence of deductive learning. Most of the deductive learners are data-driven: they solve a problem, and save the results of deductions made during the process of problem solving (a notable exception is the STATIC program by Etzioni (1991)).

Our learner saves paths (sequences of transitions) that were traversed during the search. Each path is recorded and is used in future problem solving as if it was a basic transition. We call such recorded paths *macros*. We have already discussed the reasons for choosing the general form of search space. In such search space, where there is no information about states except their successors (and heuristic values), it is not possible to generalize the macros as done in many of the existing deductive learners (Laird, Rosenbloom, & Newell, 1986; Minton, 1988). Adding any generalization mechanism would require first to eliminate the non-decomposability assumption. Composable states and generalized macros will increase significantly the complexity of the experiments and will add many factors that are hard to control.

During the search process there are many macros that can be learned. For each path with length N there are $O(N^2)$ possible macros to learn. For a search tree with depth n and branching factor k there are $O(n*k^n)$ possible macros. To make the learning feasible, we allow the learner acquiring only macros that are part of the solution path.

3. PARAMETERS THAT EFFECTS THE UTILITY OF DEDUCTIVE LEARNING

3.1. SEARCH SPACE

There are several features of the search space that may effect the utility of macro learning. The complexity of the search space is one such candidate feature. However, it is not clear how the complexity of the search space should be quantified. We would like to know how beneficial macro learning is as the space becomes harder to search. In the context of uninformed search, it should be possible to characterize spaces that are harder to search. For example, spaces with larger portions of blind alleys are harder to search. However, in the context of heuristic search, the difficulty of searching a space depends mainly on the heuristic function. A search space can have many blind alleys, but the heuristic function may be informed enough to stop the search procedure from entering such alleys. Therefore, the difficulty of heuristically searching a space depends on the combination of the space and the heuristic function.

Assuming that heuristic function is always underestimating, the least informed heuristic function is

---

[1] The estimated distance between a state (x_1,y_1) and a state (x_2,y_2) is $|x_1-x_2| + |y_1-y_2|$. In order to implement this heuristic it was necessary to represent states by ordered pairs of coordinates. This could be viewed as a violation of the non-decomposability assumption, however, all other procedures of the system refer to states as atomic entities. In principle, we could represent the heuristic function by a large table that maps symbols that represent states into distances.

the zero function. Such a function effectively turns the search into a brute force search. The most informed heuristic function is h^*, the function that always returns the cost of the minimal-cost path to the target. We have selected the average distance between h and h^* normalized by $h^*$to be the parameter that determines the search space complexity and we call it the *heuristic deviation* (in short *HD*). For each pair of states n_1,n_2),we define

$$HD_p(n_1,n_2) = \frac{h^*(n_1,n_2) - h(n_1,n_2)}{h^*(n_1,n_2)}$$

For $h^*(n_1,n_2) = 0$ (i.e., $n_1=n_2$) , $HD_p(n_1,n_2)$ is defined to be 0.

For a state space S with N distinct states we define

$$HD(S) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} HD_p(i,j)}{N^2}$$

Assuming that we do not know h*, we need to perform admissible searches between every pair of states in order to discover it. For all but very simple spaces it is not possible, thus we compute the average distance for a random sample of the space and use this as an estimate for HD.

There are two alternative methods for varying HD. We can alter the heuristic function, or we can alter the search space. We have chosen the second option.

3.2. THE SEARCH PROCEDURE

The ultimate goal of the learning process is to improve the performance of the problem solver. The characteristics of the problem solver have a major effect when we consider macro learning. The success of the learning process is determined by evaluating the performance of the problem solver with the newly acquired knowledge using the given evaluation criterion. The two major factors in such criteria are search efficiency and solution quality. When quality of solution is the dominant factor, then the problem solver uses an admissible search strategy like A*. When search efficiency is the dominant factor, then the problem solver uses a strategy like best-first that considers only the estimated cost of the remaining path to the solution, and ignores the quality of the solution. A* uses for evaluation a function that sums up the cost of the path from the initial state and the estimated cost of the remaining path. A variation of A*, called weighted A*, was proposed to allow adjustment of the weight given to the search efficiency and solution quality. Thus, the evaluation function of weighted A* is

$$f(n)=W\,h(n) + (1-W)\,g(n)$$

where n is a node in the search graph, h is the heuristic function that estimates the cost of the shortest path to a goal state, g is the cost of the path from the initial state to n, and W is a number between 0 and 1. When W=0.5 the procedure is equivalent to A* and when W=1 it is equivalent to best-first. We have selected W to be the parameter that will determine the search strategy during experimentation and call it W_P (for performance).

3.3. LEARNING

Since we assume that the learning is taking place off-line, and since we assume the general learning framework of learning-by-doing, the same type of parameter that was used to characterize the search during performance can be used to characterize the search during learning. To differentiate between the two we call the weight used during performance W_P and the weight used during learning W_L.

Another well known parameter that effects the learning process is the resources invested in learning. The most common parameter for measuring the learning resources is the number of learning instances. This is the parameter that will be used in the experiment. Other parameters, such as the number of nodes generated during learning or the number of macros learned may sometimes serve as better measurements of learning resources.

4. EXPERIMENTS

The experiments described below were all performed by varying one or two parameters while fixing all other parameters. Most of the experiments tested the performance of the problem solver on a test set of 100 randomly generated problems. Since the search graph is fully connected, every problem has a solution. Various performance measurements were taken during problem solving. A common measurement for the resources spent during problem solving is the number of nodes expanded. The reports of results below used a slightly different parameter: the number of generated nodes. For search spaces that stay constant, the two measurements are proportional. However, in the case of macro learning, it is possible that the number of successors of a state will grow rapidly, thereby increasing the cost of node expansion. The number of nodes expanded will remain constant in such a case - therefore it is not a good evaluator of the resources spent during the search. On the other hand, the number of generated nodes will grow accordingly, hence it is a better indicator of the resources spent during problem solving. Although there is a recent

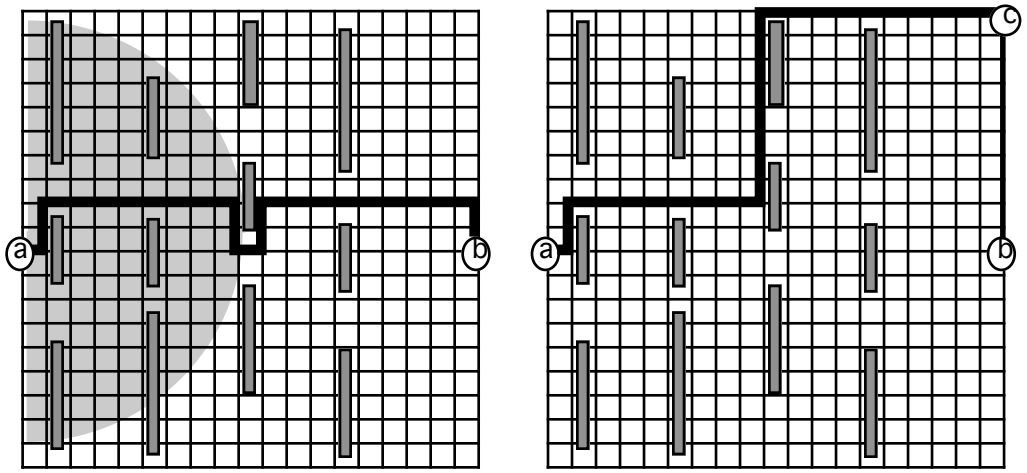

Figure 2: Macro usage in A* (a) vs. macro usage in best-first (b)

trend to report CPU time as performance measurement, we believe that it provides rather limited information because of its dependency on unrelated parameters like programming skill, language, and the machine used.

For evaluating the quality of a solution, we have measured its *optimality* : The ratio between the cost of the solution and the cost of the optimal solution. This value is available after solving the problems with A* (W_P=0.5).

4.1. THE EFFECT OF SEARCH STRATEGY DURING LEARNING AND DURING PROBLEM SOLVING ON THE UTILITY OF MACRO LEARNING

Macros have negative utility when used by a procedure that employs A* search. To understand why this is the case, assume that the current problem is to get from state A to state B in the state space illustrated in figure 2. Assume that the problem solver was lucky enough to have the macro AB at its disposal. After expanding A, B will already be in the list of generated nodes (OPEN), however A* terminates only when it is clear that the path to the goal that was found is the optimal path. Since the heuristic function is optimistic (underestimating), the search procedure will expand states that are close to the initial state (the grey half circle in figure 2a). In general, the work that A* is doing to make sure that a given path is optimal might be as demanding as the work done searching for such a path.

On the other extreme, consider best-first search. Since the speed of search is the only consideration, almost any macro that will bring the search into the vicinity of the goal state will make the search faster. However, the quality of the solution will be reduced. For example, in figure 2b, a long macro takes the procedure to state C and from there it goes directly to state B. It is clear that the search will be quite fast but the quality of solution will be reduced.

To understand the relationships between W_L (the learning strategy), W_P (the problem solving strategy) and the learning utility, we fixed the problem space, and varied the values of the two parameters to 6 different values. For each pair of values of parameters (36 all together), we performed a learning session which consisted of solving 100 test problems, solving 5 learning problems, then solving the same test again. The test set and learning set were generated randomly from the given problem space. We measured two quantities to evaluate the utility of the learning session. One quantity is the difference between the average generated nodes before the learning and after the learning. This number indicates how much was the search efficiency improved as a result of the learning session. The second quantity is the difference between the optimality values before and after learning. This number indicates how much was the solution optimality improved (or deteriorated) as a result of the learning session.

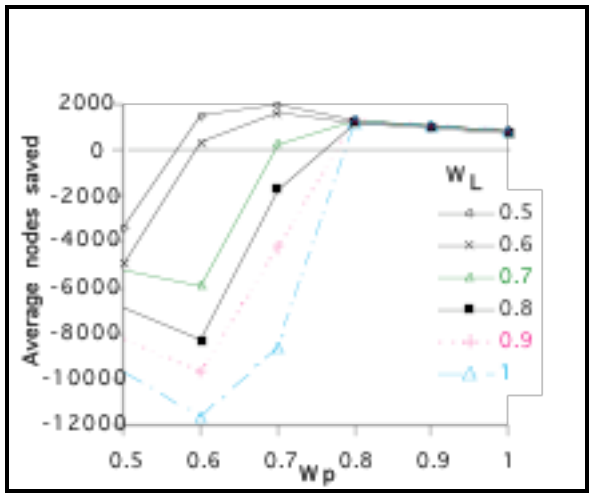

Figure 3: Improvement in performance as a function of W_P

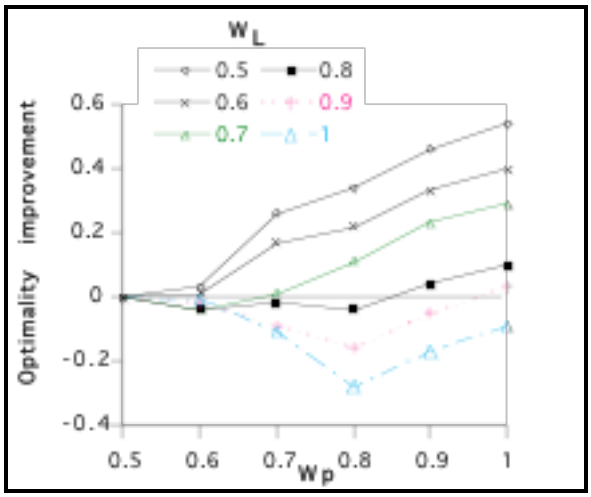

Figure 4: Improvement in optimality as a function of W_P

Each graph in Figure 3 shows the improvement in efficiency as a function of the weight W_P used during testing for a fixed W_L . There are several observations that can be made:

- When performing with A* (W_P =0.5), the utility of macros is negative, even if the macros were learned with A*.
- When performing with $W_P \geq 0.8$, macro learning was beneficial regardless of the search strategy used during learning.
- The utility of using a specific W_L depends strongly on the value of W_P . For example, with W_L =0.7, macros will be beneficial when solving problems with $W_P \geq 0.7$.
- The improvement was also calculated in percentage of the performance without learning. The graph is very similar to the one in figure 3a. The minimal improvement was -380% and the maximal was 88%.

The graphs in figure 4 evaluate the improvement in solution quality in the same way. The following observations can be made:

- While from the efficiency point of view, all values of W_L work equally well for $W_P \geq 0.8$ (for the specific search space), they are very different with regard to their effect on solution quality. As W_L decreases, the macros that are produced are of higher quality, and therefore the optimality of the solutions increases.
- Learning with high W_L (≥ 0.9) will produce such low quality macros, that the quality of the solution will decrease regardless of the W_P used.

Based on the experiments reported so far, one may conclude that it is always beneficial to learn with W_P =0.5. However, with this setting, the search during learning will be less efficient. Thus, with fixed learning resources (CPU time, or nodes generated during learning but not training examples), the learner will learn less macros and the performance will be worse than for $W_L > 0.5$.

We performed an experiment to confirm this hypothesis. We ran two learning sessions, one with W_L =0.5 and one with W_L =0.6, both with the same W_P (0.7). When we allowed both sessions to terminate after processing 5 learning problems, the performance of the problem solver with the macros learned with W_L =0.5 was twice as good as for W_L =0.6. However, when we stopped the learning

session after a fixed amount of learning resources were expended (the number of generated nodes during learning reached 5000) , the performance for 0.6 was 50% better than that for 0.5.

4.2. THE EFFECT OF SEARCH SPACE COMPLEXITY ON THE UTILITY OF MACRO LEARNING

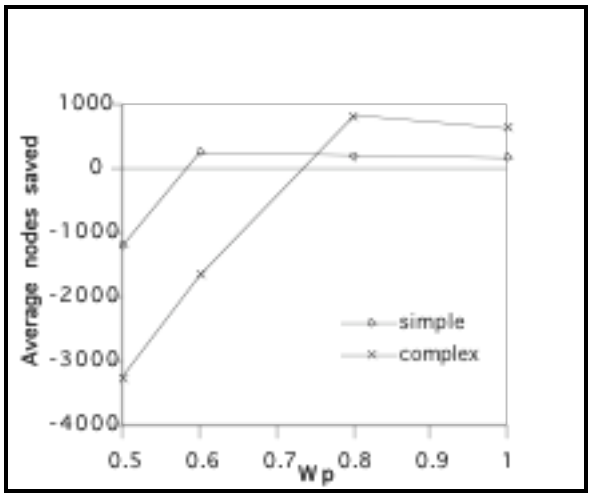

Figure 5: Improvement in performance is a simple vs. complex domain

As spaces become harder to search, the problem solver could potentially make better use of macros: subspaces that are hard to search can be skipped if the right macro is found. Thus we expect to see the utility of macro learning increasing with the complexity of the search space. However, we have already shown that the utility of the macro learning depends strongly on the search strategy. A complex space has large heuristic deviation, therefore macros will not help without being ready to give up optimality.

Figure 5 summarizes the results of the experiment where the parameters that were modified are the space complexity and the learning strategy. The y axis shows the average number of generated nodes that were saved (per problem) when macros were used. The graph shows that macro learning is harmful in the context of A* search. The negative effect is naturally more significant in the more complex domain. The graph for the simple domain crosses the 0 line earlier than the complex domain. That means that learning starts to be useful with weight greater than 0.6 for the simple domain and 0.7 for the complex domain. As we relax the optimality requirement more significantly (with a W = 0.8), macro usage in the more complex domain becomes more advantageous.

5. CONCLUSIONS

The purpose of the research described in this paper is to identify the parameters that effects deductive learning and to perform experiments systematically in order to understand the nature of those effects. The goal of this paper is to demonstrate the methodology of performing parametric experimental study of deductive learning. The example here include the study of two parameters: the point on the satisficing-optimizing scale that is used during the search carried out during problem solving time and during learning time. We showed that A*, which looks for optimal solutions, cannot benefit from macro learning but as the strategy comes closer to best-first (satisficing search), the utility of macros increases. We also demonstrated that deductive learners that learn off-line by solving training problems are sensitive to the type of search used during the learning. We showed that in general optimizing search is best for learning. It generates macros that increase the quality solutions regardless of the search method used during problem solving. It also improves the efficiency for problem solvers that require a high level of optimality. The only drawback in using optimizing search is the increase in learning resources spent.

We are aware of the fact that the results described here are not very surprising. The goal of the parametric study is not necessarily to find exciting results, but to obtain results, sometimes even previously known, in a *controlled* experimental environment.

The work described here is only part of our research plan. We are currently in the process of extensive experimentation with all the parameters described here and also with others. We also intend to test the validity of the conclusions reached during the study by repeating some of the tests in several of the commonly known search problems. We hope that such systematic experimentation will help the research community to better understand the process of deductive learning and will serve as a demonstration of the experimental methodology that should be used in machine learning research.

References

Etzioni, O. (1991). STATIC: A Problem-Space Compiler for PRODIGY. In *Proceedings of Ninth National Conference on Artificial Intelligence* (pp. 533-540).

Iba, G. A. (1989). A Heuristic Approach to the Discovery of Macro-operators. *Machine Learning, 3*, 285-317.

Kibler, D., & Langley, P. (1988). Machine Learning as an Experimental Science. In D. Sleeman (Ed.), *Proceedings of The Third European Session on Learning* (pp. 81-91). Glasgo, Scotland: Pitman.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The Anatomy of a General Learning Mechanism. *Machine Learning, 1*, 11-46.

Markovitch, S., & Scott, P. D. (1988). The Role of Forgetting in Learning. In *Proceedings of The Fifth International Conference on Machine Learning* (pp. 459-465). Ann Arbor, MI: Morgan Kaufmann.

Markovitch, S., & Scott, P. D. (1989). Utilization Filtering: a method for reducing the inherent harmfulness of deductively learned knowledge. In *Proceedings of The Eleventh International Joint Conference for Artificial Intelligence* (pp. 738-743). Detroit, Michigan.

Minton, S. (1988). *Learning Search Control Knowledge: An Explanation-Based Approach* . Boston, MA: Kluwer.

Mooney, R. (1989). The Effect of Rule Use on the Utility of Explanation-Based Learning. In *Proceedings of The Eleventh International Joint Conference for Artificial Intelligence* (pp. 725-730). Detroit, Michigan.

Pearl, J. (1984). *Heuristics : intelligent search strategies for computer problem solving* . Addison-Wesley.

Segre, A., Elkan, C., & Russell, A. (1991). A Critical Look at Experimental Evaluation of EBL. *Machine Learning, 6*, 183-195.

Simon, H. A., & Kadane, J. B. (1975). Optimal Problem-solving search: all-or-none solution. *Artificial Intelligence, 6*, 235-247.