Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA, 1991

Search Reduction in Hierarchical Problem Solving

Craig A. Knoblock[*]
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
cak@cs.cmu.edu

Abstract

It has long been recognized that hierarchical problem solving can be used to reduce search. Yet, there has been little analysis of the problem-solving method and few experimental results. This paper provides the first comprehensive analytical and empirical demonstrations of the effectiveness of hierarchical problem solving. First, the paper shows analytically that hierarchical problem solving can reduce the size of the search space from exponential to linear in the solution length and identifies a sufficient set of assumptions for such reductions in search. Second, it presents empirical results both in a domain that meets all of these assumptions as well as in domains in which these assumptions do not strictly hold. Third, the paper explores the conditions under which hierarchical problem solving will be effective in practice.

Introduction

Identifying intermediate states in a search space can be used to decompose a problem and significantly reduce search [Newell *et al.*, 1962, Minsky, 1963]. One approach to finding intermediate states is to use hierarchical problem solving [Newell and Simon, 1972, Sacerdoti, 1974], where a problem is first solved in an abstract problem space and the intermediate states in the abstract plan are used as intermediate goals to guide the search at successively more detailed abstraction levels.

While hierarchical problem solving has been used in a number of problem solvers, there has been little analysis and few empirical demonstrations of the search reductions. Both Newell et al. [1962] and Minsky [1963] present analyses that show that identifying intermediate states can reduce the depth of the search, but these analyses assume that the intermediate states are given. Korf [1987] provides an analysis of abstraction planning with macros, but his analysis assumes you are given a hierarchy of macro spaces, so that once a problem is solved in the macro space, the problem is solved. ABSTRIPS [Sacerdoti, 1974] provides the best empirical demonstration to date, but these results are in a single problem-solving domain on a small set of problems.

This paper describes hierarchical problem solving, shows that this method can reduce the size of the search space from exponential to linear in the solution length, presents the assumptions that make this reduction possible, and then describes experimental results in three different problem-solving domains. The first set of experiments provide an empirical demonstration of the exponential-to-linear search reduction in a domain that fully satisfies the stated assumptions and then explores the conditions under which hierarchical problem solving will be effective when the assumptions do not strictly hold. These experiments use the Tower of Hanoi puzzle because the highly regular structure of the problem space makes it easy to show that it satisfies the assumptions. The second set of experiments provide results in both a robot-planning and a machine-shop scheduling domain, which show that even when the assumptions of the analysis do not hold, the problem-solving method can still provide significant reductions in search.

Hierarchical Problem Solving

A *problem solver* is given a problem space, defined by the legal operators and states, and a problem, defined by an initial state and goal, and it searches for a sequence of operators that can be applied to the initial state to achieve the goal. A *hierarchical problem solver* employs a hierarchy of abstract problem spaces, called *abstraction spaces*, to focus this search process. Instead of attempting to solve a problem in the origi-

[*]The author was supported by an Air Force Laboratory Graduate Fellowship through the Human Resources Laboratory at Brooks AFB. This research was sponsored by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

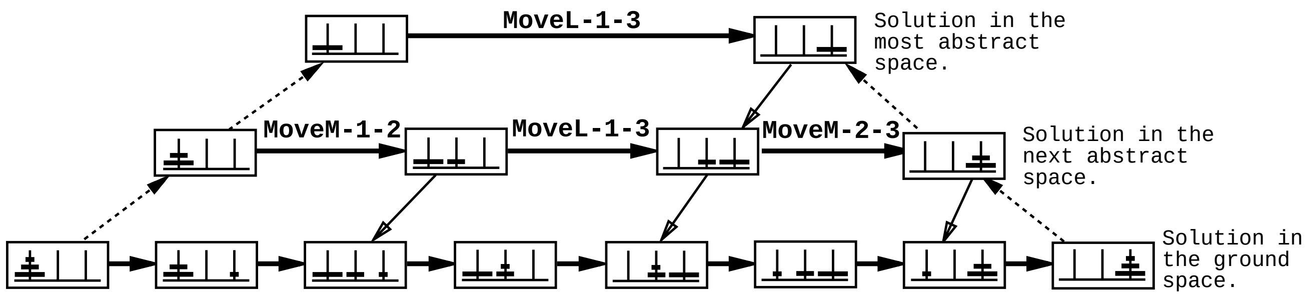
MoveL-1-3
Solution in the most abstract space.
MoveM-1-2
MoveL-1-3
MoveM-2-3
Solution in the next abstract space.
Solution in the ground space.

Figure 1: Hierarchical Problem Solving in the Tower of Hanoi

nal problem space, called the *ground space*, a hierarchical problem solver first searches for a solution in the most abstract problem space to produce a skeletal plan. This plan is then refined at successive levels in the hierarchy by inserting additional operators to produce a complete sequence of ground-level operators. This problem-solving technique was first used in GPS [Newell and Simon, 1972] and ABSTRIPS [Sacerdoti, 1974] and has since been used in a number of problem solvers, including NOAH [Sacerdoti, 1977], MOLGEN [Stefik, 1981], NONLIN [Tate, 1977], and SIPE [Wilkins, 1984].

Hierarchical problem solvers represent abstraction spaces in various ways and employ a variety of techniques for refining an abstract plan. In this paper, the language of each successive abstraction space is a subset of the previous problem spaces and the operators and states in an abstraction space correspond to one or more operators or states in the more detailed problem spaces. Given a hierarchy of abstraction spaces, hierarchical problem solving proceeds as follows. First, the problem solver maps the given problem into the most abstract space by deleting literals from the initial state and goal that are not relevant to the abstract space. Next, the problem solver finds a solution that solves the abstract problem. Each of the intermediate states in the abstract plan serve as goals for the subproblems at the next level in the abstraction hierarchy. The problem solver then solves each of the intermediate subproblems using the final state of one subproblem as the initial state for the next subproblem. The intermediate states of the plan at this new level then serve as goals for the subproblems at the next level, and the process is repeated until the plan is refined into the ground space. This approach to hierarchical problem solving is formally defined in [Knoblock, 1991].

Consider an abstraction hierarchy for the three-disk Tower of Hanoi, where the most abstract space contains only the largest disk, the next abstraction space contains the largest and medium-sized disk, and the ground space contains all three disks. This hierarchy can be used for problem solving as shown in Figure 1. First, the initial and goal states are mapped into the abstract space by dropping the smaller disks. Next, the problem is solved in the most abstract space, which simply requires a one step plan that moves the largest disk (`diskL`) from `peg1` to `peg3`. This creates two subproblems at the next level of abstraction, where the first subproblem is to reach the state where the abstract operator can be applied, and the second subproblem is to reach the goal state. After solving these subproblems, the problem solver repeats the process at the next level and produces a plan that solves the original problem.

Analysis of the Search Reduction

This section presents a complexity analysis of hierarchical problem solving, which shows that, under an ideal decomposition of a problem, hierarchical problem solving reduces the worst-case complexity of the search from exponential to linear in the solution length. Since the size of the search spaces are potentially infinite, the analysis assumes the use of an admissible search procedure (e.g., depth-first iterative-deepening [Korf, 1985]), which is bounded by the length of the shortest solution.

The analysis is similar to the analysis of abstraction planning with macros by Korf [1987]. Korf showed that using a hierarchy of macros can reduce an exponential search to a linear one. However, Korf's analysis applies to abstraction planning with macros and not to hierarchical problem solving because it makes several assumptions that do not hold for the latter. The most significant assumption of the analysis is that when the abstract problem is solved, the original problem is solved. Using hierarchical problem solving, once a problem has been solved in the abstract space, the abstract solution must still be refined into a solution in the ground space.

Single-Level Problem Solving

For single-level problem solving, if a problem has a solution of length l and the search space has a branching factor b, then in the worst-case the size of the search space is $\sum_{i=1}^{l} b^i$. Thus, the worst-case complexity of this problem is $O(b^l)$.

Two-Level Problem Solving

Let k be the ratio of the solution length in the ground space to the solution length in the abstract space. Thus, $\frac{l}{k}$ is the solution length in the abstract space. Since each operator in the abstract space corresponds to one or more operators in the ground space, the branching factor of the abstract space is bounded by the branching factor of the ground space, b. The size of the search tree in the abstract space is $\sum_{i=1}^{l/k} b^i$, which is $O(b^{\frac{l}{k}})$. In addition, the analysis must include the use of this abstract solution to solve the original problem.

The abstract solution defines $\frac{l}{k}$ subproblems. The size of each problem is the number of steps (solution length) in the ground space required to transform an initial state S_i into a goal state S_{i+1}, which is represented as $d(S_i, S_{i+1})$.

$$\sum_{i=1}^{d(S_0,S_1)} b^i + \sum_{i=1}^{d(S_1,S_2)} b^i + \cdots + \sum_{i=1}^{d(S_{\frac{l}{k}-1},S_{\frac{l}{k}})} b^i \qquad (1)$$

which is $O(\frac{l}{k} b^{d_{\max}})$, where

$$d_{\max} \equiv \max_{0 \le i \le \frac{l}{k}-1} d(S_i, S_{i+1}) \qquad (2)$$

In the ideal case, the abstract solution will divide the problem into subproblems of equal size, and the length of the final solution using abstraction will equal the length of the solution without abstraction. In this case, the abstract solution divides the problem into $\frac{l}{k}$ subproblems of length k.

$$b^{d_{\max}} = b^{\frac{l}{l/k}} = b^k \qquad (3)$$

Assuming that the planner can first solve the abstract problem and then solve each of the problems in the ground space without backtracking across problems, then the size of the space searched in the worst case is the sum of the search spaces for each of the problems.

$$\sum_{i=1}^{\frac{l}{k}} b^i + \frac{l}{k}\sum_{i=1}^{k} b^i \qquad (4)$$

The complexity of this search is: $O(b^{\frac{l}{k}} + \frac{l}{k}b^k)$. The high-order term is minimized when $\frac{l}{k} = k$, which occurs when $k = \sqrt{l}$. Thus, when $k = \sqrt{l}$, the complexity is $O(\sqrt{l}\ b^{\sqrt{l}})$, compared to the original complexity of $O(b^l)$.

Multi-Level Problem Solving

Korf [1987] showed that a hierarchy of macro spaces can reduce the expected search time from $O(s)$ to $O(\log s)$, where s is the size of the search space. This section proves an analogous result – that multi-level hierarchical problem solving can reduce the size of the search space for a problem of length l from $O(b^l)$ to $O(l)$.

In general, the size of the search space with n levels (where the ratio between the levels is k) is:

$$\sum_{i=1}^{\frac{l}{k^{n-1}}} b^i + \frac{l}{k^{n-1}}\sum_{i=1}^{k} b^i + \frac{l}{k^{n-2}}\sum_{i=1}^{k} b^i + \cdots + \frac{l}{k}\sum_{i=1}^{k} b^i \qquad (5)$$

The first term in the formula accounts for the search in the most abstract space. Each successive term accounts for the search in successive abstraction spaces. Thus, after solving the first problem, there are l/k^{n-1} subproblems that will have to be solved at the next level. Each of these problems are of size k, since k is the ratio of the solution lengths between adjacent abstraction levels. At the next level there are l/k^{n-2} subproblems ($k \times l/k^{n-1}$) each of size k, and so on. In the final level there are $\frac{l}{k}$ subproblems each of size k. The final solution will therefore be of length $\frac{l}{k}k = l$.

The maximum reduction in search can be obtained by setting the number of levels n to $\log_k(l)$, where the base of the logarithm is the ratio between levels. Substituting $\log_k(l)$ for n in Formula 5 above produces the following formula:

$$\sum_{i=1}^{k} b^i + k\sum_{i=1}^{k} b^i + k^2\sum_{i=1}^{k} b^i + \cdots + k^{\log_k(l)-1}\sum_{i=1}^{k} b^i \qquad (6)$$

From Formula 6, it follows that the complexity of the search is:

$$O((1 + k + k^2 + \cdots + k^{\log_k(l)-1})b^k). \qquad (7)$$

The standard summation formula for a finite geometric series with n terms, where each term increases by a factor of k, is:

$$1 + k + k^2 + \cdots + k^n = \frac{k^{n+1}-1}{k-1}. \qquad (8)$$

Using this equation to simplify Formula 7, it follows that the complexity is:

$$O(\frac{k^{\log_k(l)}-1}{k-1}b^k) = O(\frac{l-1}{k-1}b^k). \qquad (9)$$

Since b and k are assumed to be constant for a given problem space and abstraction hierarchy, the complexity of the entire search space is $O(l)$.

Assumptions of the Analysis

The analysis above makes the following assumptions:

1. *The number of abstraction levels is* $\log_k$ *of the solution length.* Thus, the number of abstraction levels must increase with the size of the problems.
2. *The ratio between levels is the base of the logarithm,* k.
3. *The problem is decomposed into subproblems that are all of equal size.* If all the other assumptions hold, the complexity of the search will be the complexity of the largest subproblem in the search.

4. *The hierarchical planner produces the shortest solution.* The analysis holds as long as the length of the final solution is linear in the length of the optimal solution.

5. *There is only backtracking within a subproblem.* This requires that a problem can be decomposed such that there is no backtracking across abstraction levels or across subproblems within an abstraction level.

The assumptions above are sufficient to produce an exponential-to-linear reduction in the size of the search space. The essence of the assumptions is that the abstraction divides the problem into $O(l)$ constant size subproblems that can be solved in order.

Consider the abstraction of the Tower of Hanoi described in the previous section. It is ideal in the sense that it meets all of the assumptions listed above. First, the number of abstraction levels is $O(\log_2(l))$. For an n-disk problem the solution length l is $2^n - 1$, and the number of abstraction levels is n, which is $O(\log_2(l))$. Second, the ratio between the levels is the base of the logarithm, which is 2. Third, these subproblems are effectively all of size one, since each subproblem requires inserting one additional step to move the disk added at that abstraction level. Fourth, using an admissible search strategy, the hierarchical problem solver produces the shortest solution. Fifth, the only backtracking necessary to solve the problem is within a subproblem.

Since these assumptions are sufficient to reduce the size of the search space from exponential to linear in the length of the solution, it follows that hierarchical problem solving produces such a reduction for the Tower of Hanoi. While these assumptions hold in this domain, they will not hold in all problem domains. Yet, hierarchical problem solving can still provide significant reductions in search. The next section explores the search reduction in the Tower of Hanoi in practice, and the section following that explores the search reduction in more complex domains where many of the assumptions do not strictly hold.

Search Reduction: Theory vs. Practice

The previous section showed analytically that hierarchical problem solving can produce an exponential-to-linear reduction in the size of the search space. This section provides empirical confirmation of this result and then explores the conditions under which hierarchical problem solving will be effective in practice. The experiments were run on the Tower of Hanoi both with and without using the abstraction hierarchy described in the preceding sections. The abstractions were automatically generated by the ALPINE system [Knoblock, 1990] and then used in a hierarchical version of the PRODIGY problem solver [Minton *et al.*, 1989].

To evaluate empirically the use of hierarchical problem solving in the Tower of Hanoi, PRODIGY was run both with and without the abstractions using a depth-first iterative-deepening search, a depth-first search, and a depth-first search on a slightly modified version of the problem. The experiments compare the CPU time required to solve problems that range from one to seven disks. The graphs below measure the problem size in terms of the optimal solution length, not the number of disks, since the solution to a problem with n disks is twice as long as the solution to a problem with $n - 1$ disks. For example, the solution to the six-disk problem requires 63 steps and the solution to the seven-disk problem requires 127 steps.

Figure 2 compares PRODIGY with and without hierarchical problem solving using depth-first iterative-deepening to solve the problems and subproblems. As the analytical results predict, the use of abstraction with an admissible search procedure produces an exponential reduction in the amount of search. The results are plotted with the problem size along the x-axis and the number of nodes searched along the y-axis. With abstraction the search is linear in the problem size and without abstraction the search is exponential. In the Tower of Hanoi, the use of an admissible search produces optimal (shortest) solutions both with and without abstraction.

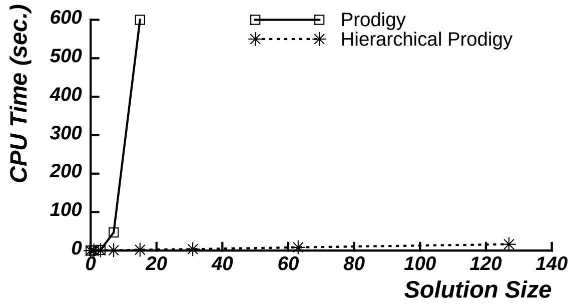

Figure 2: Comparison using depth-first iterative-deepening in the Tower of Hanoi.

Admissible search procedures such as breadth-first search or depth-first iterative-deepening are guaranteed to produce the shortest solution[1] and to do so usually requires searching most of the search space. However, these methods are impractical in more complex problems, so this section also examines the use of hierarchical problem solving with a nonadmissible search procedure. Figure 3 compares the CPU time for problem solving with and without abstraction using depth-first search. As the graph shows, the use of abstraction produces only a modest reduction in search. This is because, using depth-first search, neither configuration is performing much search. When the problem solver makes a mistake it simply proceeds adding steps to undo the mistakes. Thus, the number of nodes searched by each configuration is roughly linear in the

[1] Due to the decomposition of a problem, an admissible search is not guaranteed to produce the optimal solutions for hierarchical problem solving.

length of the solutions found. Problem solving with abstraction performed better because the abstraction provides some guidance on which goals to work on first and thus produces shorter solutions by avoiding some unnecessary steps.

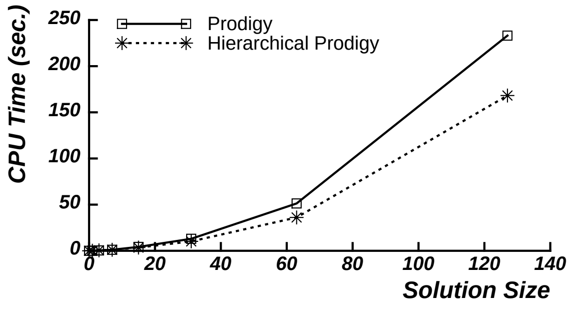

Figure 3: Comparison using depth-first search in the Tower of Hanoi.

The small difference between depth-first search with and without using abstraction is largely due to the fact that the problems can be solved with relatively little backtracking. To illustrate this point, consider a variant of the Tower of Hanoi problem that has the additional restriction that no disk can be moved twice in a row [Anzai and Simon, 1979]. By imposing additional structure on the domain, the problem solver is forced to do more backtracking. Figure 4 compares the CPU time used by the two configurations on this variant of the domain. This small amount of additional structure enables the hierarchical problem solver to produce optimal solutions in linear time, while PRODIGY produces suboptimal solutions that requires significantly more problem-solving time.

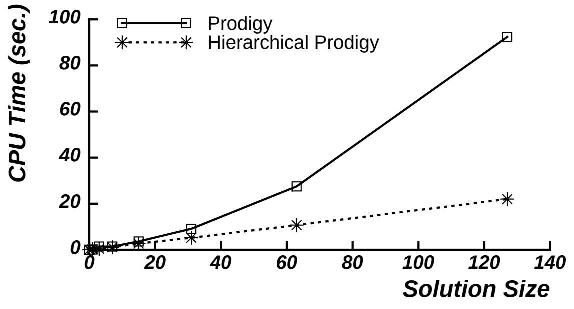

Figure 4: Comparison using depth-first search in a variant of the Tower of Hanoi.

The use of abstraction produces large search reductions over problem solving without abstraction only when a large portion of the search space must be explored to find a solution. In addition, the problem solver can sometimes trade off solution quality for solution time by producing longer solutions rather than searching for better ones. The Tower of Hanoi is perhaps a bit unusual in that the structure of the search space allows the problem solver to undo its mistakes by simply inserting additional steps. In domains that are more constrained, the problem solver would be forced to backtrack and search a fairly large portion of the search space to find a solution. To demonstrate this claim, the next section presents results in two more complex problem-solving domains, where it would be infeasible to use an admissible search.

Experimental Results

This section describes the results of hierarchical problem solving in PRODIGY in two problem-solving domains: an extended version of the STRIPS robot-planning domain and a machine-shop scheduling domain. These domains were described in [Minton, 1988], where they were used to evaluate the effectiveness of the explanation-based learning module in PRODIGY. The abstraction hierarchies used in these experiments were automatically generated by ALPINE and are fully described in [Knoblock, 1991].

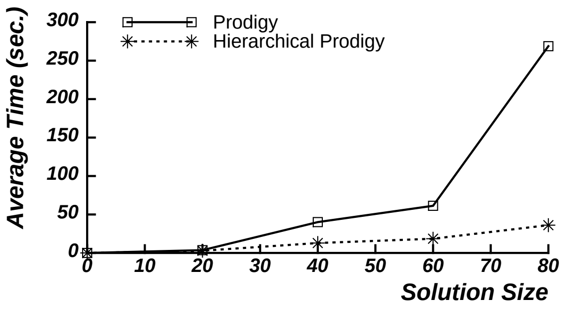

Figure 5: Comparison in the robot-planning domain.

Figures 5 and 6 compare the average CPU time on problems of increasing size both with and without using hierarchical problem solving. Both problem domains were tested on large sets of randomly generated problems (between 250 and 400 problems). Some of the problems could not be solved by PRODIGY within 10 minutes of CPU time. These problems are included in the graphs since including problems that exceed the time bound underestimates the average, but provides a better indication of overall performance. The graphs show that on simple problems PRODIGY performs about the same as hierarchical PRODIGY, but as the problems become harder the use of hierarchical problem solving clearly pays off. In addition, hierarchical problem solving produces solutions that are about 10% shorter than PRODIGY.

Unlike the Tower of Hanoi, these two problem-solving domains do not satisfy the assumptions described in the analysis. There is backtracking both across subproblems and across abstraction levels, the solutions are sometimes suboptimal, and the problems are not partitioned into equal size subproblems. Despite this, the use of hierarchical problem solving in these domains still produces significant reductions in

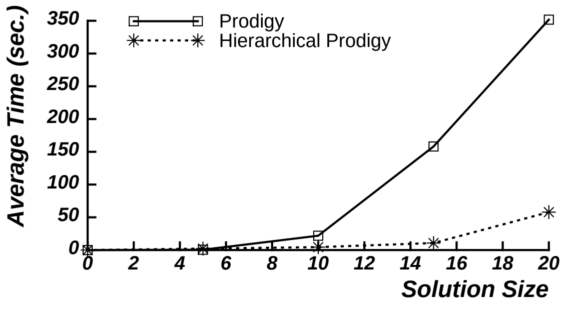

Figure 6: Comparison in the machine-shop domain.

search. On the harder sets of problems, the graphs show that even the hierarchical problem solver begins to search more. In these domains, this can be attributed to the fact that ALPINE does not currently find the best abstraction hierarchies for these problems, but this is a limitation of ALPINE and not of the hierarchical problem solver.

Conclusion

While hierarchical problem solving has long been claimed to be an effective technique for reducing search, there has been no detailed analysis and few empirical results. This paper presented a method for hierarchical problem solving, showed that this method can produce an exponential-to-linear reduction in the search space, and identified the assumptions under which such a reduction is possible. In addition, the paper provided empirical results that show that hierarchical problem solving can reduce search in practice, even when the set of assumptions does not strictly hold.

There are several interesting conclusions that one can draw from the experiments. First, the degree to which abstraction reduces search depends on the portion of the ground-level search space that is explored without using hierarchical problem solving. Thus, the more backtracking in a problem, the more benefit provided by the use of abstraction. Second, with a non-admissible search procedure the use of abstraction will tend to produce shorter solutions since the abstractions focus the problem solver on the parts of the problem that should be solved first. Third, although many domains lack the highly regular structure of the Tower of Hanoi, hierarchical problem solving can still provide significant reductions in search.

Acknowledgements

I am grateful to my advisor, Jaime Carbonell, for his guidance and support. I would also like to thank Jane Hsu, Paul Rosenbloom, and Manuela Veloso for their detailed comments on the analysis, as well as Claire Bono, Oren Etzioni, and Qiang Yang for their helpful comments and suggestions on earlier drafts of this paper.

References

[Anzai and Simon, 1979] Yuichiro Anzai and Herbert A. Simon. The theory of learning by doing. *Psychological Review*, 86:124–140, 1979.

[Knoblock, 1990] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Tech. Report CMU-CS-91-120.

[Korf, 1985] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[Korf, 1987] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[Minsky, 1963] Marvin Minsky. Steps toward artificial intelligence. In Edward A. Feigenbaum, editor, *Computers and Thought*, pages 406–450. McGraw-Hill, New York, NY, 1963.

[Minton *et al.*, 1989] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40(1-3):63–118, 1989.

[Minton, 1988] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Computer Science Department, Carnegie Mellon University, 1988.

[Newell and Simon, 1972] Allen Newell and Herbert A. Simon. *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Newell *et al.*, 1962] Allen Newell, J. C. Shaw, and Herbert A. Simon. The processes of creative thinking. In *Contemporary Approaches to Creative Thinking*, pages 63–119. Atherton Press, New York, 1962.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Sacerdoti, 1977] Earl D. Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, New York, NY, 1977.

[Stefik, 1981] Mark Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16(2):111–140, 1981.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–900, Cambridge, MA, 1977.

[Wilkins, 1984] David E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22(3):269–301, 1984.