The Expected Value of Hierarchical Problem-Solving*

Fahiem Bacchus and **Qiang Yang**
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Abstract

In the best case using an abstraction hierarchy in problem-solving can yield an exponential speed-up in search efficiency. Such a speed-up is predicted by various analytical models developed in the literature, and efficiency gains of this order have been confirmed empirically. However, these models assume that the *Downward Refinement Property* (DRP) holds. When this property holds, backtracking never need occur across abstraction levels. When it fails, search may have to consider many different abstract solutions before finding one that can be refined to a concrete solution. In this paper we provide an analysis of the expected search complexity without assuming the DRP. We find that our model predicts a phase boundary where abstraction provides no benefit: if the probability that an abstract solution can be refined is very low or very high, search with abstraction yields significant speed up. However, in the phase boundary area where the probability takes on an intermediate value search efficiency is not necessarily improved. The phenomenon of a phase boundary where search is hardest agrees with recent empirical studies of Cheeseman et al. [CKT91].

Introduction

In this paper we examine the benefits of hierarchical problem-solving. Hierarchical problem-solving is accomplished by first searching for an abstract solution to the problem and then using the intermediate states of the abstract solution as intermediate goals to decompose the search for the non-abstract solution. This technique has been used in a number of problem-solvers in AI [NS72, Sac74, Sac77, Ste81, Tat77, Wil84].

*This work is supported by grants from the Natural Science and Engineering Council of Canada and by the Institute for Robotics and Intelligent Systems. The authors' e-mail addresses are `fbacchus@logos.waterloo.ca` and `qyang@logos.waterloo.ca`.

It has long been known that the identification of intermediate states which decompose a problem can significantly reduce search [NSS62, Min63]. However, the analysis of the benefit yielded by decomposition, provided in these works, ignores the cost of finding the intermediate states. In hierarchical problem-solving these states are found by searching in an abstract version of the problem-space. The abstract space is smaller and hence the benefit gained by decomposing the non-abstract space often outweights the cost of searching this space. Empirical evidence of the net benefit of the hierarchical approach has been provided by ABSTRIPS [Sac74] and by the work of Newell and Simon [NS72]. However, only small problems and limited domains were considered by these works.

Korf [Kor85] has provided an analysis of the benefits of using macro operators as the abstraction device. With this type of abstraction, however, once we find a solution in the abstract space (the space generated by the macro operators) we have a non-abstract solution: no further search is required. Nevertheless, Korf's analysis can be viewed as demonstrating that searching for an abstract solution is significantly more efficient that searching for a non-abstract solution.

Knoblock's analysis of hierarchical problem-solving [Kno91] is the most detailed to date, and has had a significant influence on this work. However, his analysis assumes that backtracking does not occur across abstraction levels: once an abstract solution is found we need never search for another one. Hence, Knoblock's work can be viewed as demonstrating that searching the decomposed non-abstract space plus searching the abstract space *once* yields a significant net benefit over searching the non-abstract space.

In previous work [BY91] we have identified this assumption as an important property of an abstraction hierarchy, and have termed it the *downward refinement property* (DRP). Formally, this property holds when *ev-*

ery abstract solution can be refined in a useful manner[1] to the next lower level of abstraction. This implies that an abstract solution can always be refined to a concrete solution *without backtracking across abstraction levels*, given that a concrete-level solution to the planning problem exists.

When the DRP fails the planner may expend search effort trying to refine a particular abstract solution before discovering that it is unrefineable. This would cause backtrack in the abstraction hierarchy to find an alternate abstract solution. Search would then continue by trying to refine this new abstract solution. Clearly, if such backtracking occurs frequently the overhead of searching the abstraction hierarchy could overwhelm the benefits of using abstraction. In fact, experiments with ABSTRIPS and ABTWEAK [YT90] have shown that abstraction only increases search efficiency in hierarchies where the probability is high that an abstract solution is refineable (i.e., where we do not have to do much backtracking in the abstraction hierarchy). In hierarchies where this is not the case, using abstraction can in fact decrease the efficiency of the planner.[2]

In order to understand this phenomenon more thoroughly we provide an analytical model of search complexity[3] as a function of this probability, i.e., the probability that an abstract solution can be refined. This also provides a more realistic analysis of the benefits of abstraction in problem-solving: unlike previous models it takes the important factor of search through the abstraction hierarchy into consideration.

Our analysis demonstrates the existence of two qualitatively distinct cases. When we have the DRP or when the probability that an abstract solution can be refined is low, search complexity does not depend on the shape of the abstraction space, and in fact it can be made linear if the number of levels of abstraction can be made large enough. On the other hand, in the middle region, where the DRP fails and the probability of refinement is not that small, search complexity depends on both the number of levels of abstraction and on the branching factor in the abstraction space. In this region increasing the number of levels of abstraction is not always a useful option as that also increases search complexity.

An additional contribution of this work is that it provides an analytical model that supports the recent empirical results reported by Cheeseman et al. [CKT91]. At the extremes where most abstract solutions can be refined and where very few can, search is relatively easy. In the former case backtracking is minimized, while in the latter case it does not require much work to recognize that you are on the wrong path, i.e., backtracking is cheap. The middle region, however, represents a phase boundary where a larger proportion of hard search problems lie: average search complexity rises in this region. Here, a significant fraction of the abstract solutions are unrefineable, and it can take a great deal of work to detect that you are on a bad path.

In the sequel we will first present the basic problem-solving framework under which we are working, and identify the assumptions which make the analysis tractable. We then present the details of our analysis. From the model we are able to generate various predictions and we discuss those next. Finally, we close with a discussion of the implications of the work and some conclusions.

The Problem-Solving Framework

The problem-space is defined by a collection of states and operators which map between the states. A problem consists of an initial state and a goal state, and it is solved by searching for a sequence of operators whose composition will map the initial state to the goal state. In hierarchical problem-solving an abstract version of the original, *ground* or *concrete*, problem-space is used. The abstract version is generated via some reduction or generalization of the operators or states in the ground space. For example, in ABSTRIPS the operators are generalized by dropping some of their preconditions; this has the effect of increasing the domain of the function they define on the states.

A hierarchical problem-solver first searches the abstract space for a solution. However, this solution will no longer be correct when we move to a lower level of abstraction; instead it can only serve as a skeletal plan for the lower level. A correct solution at the lower level is generated by *refineing* the abstract plan, and this is accomplished by inserting additional operators between the operators in the abstract plan. If we have m operators in the abstract plan, refinement to the next lower level can be viewed as solving m "gap" subproblems. Solving the gaps amounts to finding new sequences of operators which when placed between the operators of

[1] There is a formal characterization of "useful" which ensures that the work done at the abstract level is not undone during refinement. Such refinements are termed *monotonic*. See [KTY91] and [BY91] for more details.

[2] This effect is in part due to the additional (constant factor) overhead involved in using abstraction.

[3] There have been other analytical models of search complexity presented in the literature, e.g., [KP83, MP91]. However, these works have addressed fundamentally different search problems. For example, the works cited consider the problem of searching for an optimal path in an infinite binary tree with branches of cost 1 and 0. Search through a hierarchy of abstraction spaces, considered here, cannot be mapped to this model.

the abstract plan generate a correct solution at the lower level.

The Analytical Model

The Tree of Abstract Plans

The total search space explored by a hierarchical problem-solver can be viewed as a tree generated by the abstraction hierarchy. In this tree each node at level i represents a complete i-th level abstract plan. The children of a node represent all of the different refinements of that plan at the next (lower) level of abstraction. The leaf nodes are complete concrete-level plans. The task in searching through the abstraction space is to find a path from the root down to a leaf node representing a correct concrete-level solution. Each node on the path must be a legal i-th level abstract plan to the problem at hand and must be a refinement of the $i+1$-level abstract plan represented by its parent. The work in searching this tree comes from the work required to find the plan at each node and will depend on the number and depth of the nodes the search examines.

The root represents a special length one solution to every problem: a universal plan. Its presence is simply a technical convenience. The levels of the tree are numbered $\{n, \ldots, 0\}$ with the root being at level n and the leaves at level 0. Hence, discounting the universal plan at level n, our abstraction hierarchy has n levels. To make our analysis useful we make some additional assumptions.

First, we assume that the abstraction hierarchy is regular. In particular, we assume that it takes approximately k new operators to solve every gap subproblem where k is constant across abstraction levels. Refineing a solution to the next level amounts to solving a gap subproblem between every pair of operators; hence the refined solution will be k times longer. Since the root is a solution of length 1, this means that the solutions at level i are of length k^{n-i}, and that the concrete-level solution is of length k^n, which we also denote by ℓ.

As this assumption degenerates the value of abstraction degenerates. If we end up having gap subproblems which require solutions of length $O(\ell)$ instead of $O(k) = O(\ell^{1/n})$, then solving them will require search of $O(b^\ell)$ where b is the branching factor generated by the operators in the ground space.[4] This is no better than search without abstraction.

Second, we assume that the individual gap subproblems can be solved without significant interaction. If, say, r gap subproblems interact we will have to search for a plan that solves all of them simultaneously. Such a plan would be of length $O(rk)$ and would require $O(b^{rk})$ search. As rk approaches ℓ we once again degenerate to search complexity of $O(b^\ell)$ where abstraction yields no benefits.

Our two assumptions, then, are basic assumptions required before the abstraction hierarchy yields any interesting behavior at all. When these assumptions fail the abstraction hierarchy is simply not decomposing the problem effectively. Knoblock [Kno91] also relies on these assumptions, but his assumption of independent subproblems is phrased as an assumption that backtracking only occurs within a subproblem. This is significantly stronger as it also prohibits backtracking across abstraction levels.

The tree of abstract plans will have a branching factor that, in general, will vary from node to node. This branching factor is the number of $i-1$ level refinements possible for a given i level solution, i.e., the number of children a node at level i has. Let the maximum of these branching factor be B. For simplicity we will use B as the branching factor for all nodes in the tree. Note, B has no straightforward relationship with b the branching factor generated by the operators.

The Probability of Refinement

If a hierarchy has the DRP then every solution at abstraction level i can be refined to a solution at abstract level $i-1$. A reasonable way in examine the behavior of hierarchies in which the DRP fails is to assign a probability, p, to the event that a given i-th level solution can be refined to level $i-1$. DRP now corresponds to the case $p = 1$. If $p = 1$ we need never reconsider the initial part of a path of good solutions. The DRP guarantees we can extend the path down to level 0. If $p < 1$, however, we might build a path of correct solutions from the root down to a node at level i, and then find, upon examining all of its children, that it is not refineable to the next level. This will force a backtrack to the penultimate node at level $i + 1$ to find an alternate level i solution, one which is refineable. This may cause further backtrack to level $i + 2$, or search may progress to lower levels before backtracking occurs again.

We are interested in the complexity of search when a ground-level solution exists. In this case it follows from the upward solution property [Ten89] that there will be at least one path of correct solutions in the tree from the root to a leaf node. Our task, then, is to explore the average case complexity of search in abstraction hierarchies in which (1) the probability that a given node in the abstraction search tree can be refined is p, and (2) there is at least one *good path*, i.e., a path of good nodes, from the root to a leaf in the tree.

[4] The branching factors of the abstract spaces are lower, but we can use b as an upper bound.

Average case complexity can be found by considering randomly generated abstraction trees. Our tree has a constant branching factor B and height $n+1$. Hence, it has $N = \frac{B^{n+1}-1}{B-1}$ nodes. A random tree is generated by labeling each node independently as being refineable (*good*) with probability p, or not refineable (*bad*) with probability $1-p$. Each of the 2^N distinct trees that can be generated by this process has probability $p^g(1-p)^{N-g}$, where g is the number of good nodes. Some of these trees will not contain a good path from the root to a leaf. We remove these trees, and renormalize the probabilities of the remaining trees so that they sum to 1. That is, we take the conditional probability.

One other piece of notation we use is $b(K, N, P)$ to denote the binomial distribution, i.e., the probability of K successes in N independent Bernoulli trials each with probability P of success.

Analytic Forms

Now we present the analytic forms which result from an analysis of the above model. The reader is referred to our full report for the proofs of the following results [BY94].

1) Let $\mathsf{NodeWork}(i)$ be the amount of work required to refine a node at level i. At level n we have one subproblem to solve which requires $O(b^k)$ computation. At level $n-1$ the nodes are abstract solutions of length k, resulting in k subproblems each requiring $O(b^k)$ computation. This trend continues to level 1, but at level 0 the solutions are concrete and do not need to be refined. Hence, we have $\mathsf{NodeWork}(i) = k^{n-i}b^k$ and $\mathsf{NodeWork}(0) = 0$.

2) Let $F(i)$ be the probability that a random subtree rooted at level i fails to contain a good path from its root to a leaf. A subtree can fail to contain a good path in two exclusive ways: (a) the root could be a bad node or (b) the root could be good but somewhere among its descendents all the good paths terminate before reaching level 0.

The second case can be analyzed using the theory of branching processes [Fel68, AN72]. If the root is good it initiates a branching process where it might have some number of good children and they in turn might have some number of good children and so on. We can consider the production of bad children as points were the process terminates. The number of good children of the root is binomially distributed: $b(m, B, p)$ is the probability of having m good children, and its generating function is $G(s) = (q+ps)^B$,[5] where $q = 1-p$. Let $G_1(s) = G(s)$ and $G_j = G_{j-1}(G(s))$, i.e., the j-th iterate of $G(s)$.

From the theory of branching processes it is known that the probability that there are no path of i good nodes from the root is $G_i(0)$. For example, the probability of there being no paths of 2 good nodes from the root (i.e., the probability of no good child having a good child) is $G_2(0) = G(G(0)) = G(q^B) = (q+pq^B)^B$.

Putting (a) and (b) together we obtain: $F(i) = q + p[G_i(0)]$, and we can compute $F(i)$ directly for any value of i and p. From this result and known results about the asymptotic behavior of branching processes we can identify three regions of importance: when $p \le 1/B$ we have $\lim_{i\to\infty} F(i) = 1$; when $1/B < p < 1$ we have $0 < \lim_{i\to\infty} F(i) < 1$ with the value of the limit decreasing as p increases; and when $p = 1$ we have $\forall i. F(i) = 0$.

3) Let $\mathsf{BadTreeWork}(i)$ be the expected amount of computation required to search a subtree with root at level i that does not contain a good path. To ensure that such a tree is a dead end we have to search until we have exhausted all candidate good paths. We always have to expand the root which is at level i and hence requires $\mathsf{NodeWork}(i) = k^{n-i}b^k$ computation. With probability p the root is refineable and we will then have to examine all of the B subtrees under the root, all of which must be bad (otherwise the initial tree would not be bad). This process must stop by level 1, as if a node at level 1 is good this means that it can be refined to a good ground-level solution and the initial tree would not be bad. Hence, $\mathsf{BadTreeWork}(0) = 0$, and we obtain the recurrence: $\mathsf{BadTreeWork}(i) = k^{n-i}b^k + pB(\mathsf{BadTreeWork}(i-1))$. By expanding the first few terms of this recurrence we can find a general expression:

$$\mathsf{BadTreeWork}(i) = b^k k^{n-i}\frac{(pBk)^i - 1}{pBk - 1}. \qquad (1)$$

4) Let $\mathsf{GoodTreeWork}(i)$ be the expected amount of computation required to search a *good subtree* with root at level i, i.e., a subtree which contains at least one good path from its root to a leaf. Our ultimate aim is to analyze $\mathsf{GoodTreeWork}(n)$. To examine a good tree we have to expand its root node. Then we must search the subtrees under the root, looking for a good subtree rooted at the next level. Once we find such a subtree we never need backtrack out of it. (There may however, be any amount of backtracking involved while searching the bad subtrees encountered before we find a good subtree.)

The root has B children, and hence between 1 and B good subtrees under it. Let m be the number of

[5] When this generating function is expanded as a power series in s the coefficient of s^m is equal to the probability of m good (refineable) children among the B offspring, i.e., $b(m, B, p)$.

good subtrees under the root. The probability of m taking any particular value is $b(m, B, 1-F(i-1))$: each subtree can be viewed to be the result of a Bernoulli trial where the probability of failure (a bad subtree) is $F(i-1)$. However, we also know that the case $m = 0$ is impossible, and must renormalize the probabilities by dividing them by $1 - b(0, B, 1 - F(i-1))$. If there are in fact m good subtrees, then by it can be proved [BY94] that on average we will have to search $(B - m)/(m + 1)$ bad subtrees before finding a good subtree. The expected number of bad subtrees that must be searched can then be computed by summing the average number of trees for each values of m times the probability of that value of m holding.

The observations above can be put together to yield a recurrence which can be simplified to the following form.

$$\mathsf{GoodTreeWork}(i) = k^{n-i} b^k \left(\frac{k^i - 1}{k - 1}\right) + \sum_{j=1}^{i-1} \mathsf{BadTreeWork}(j)\Gamma(j). \tag{2}$$

In this equation $\Gamma(i)$ represents the average number of bad subtrees we need to examine at level i. A closed form for $\Gamma(i)$ involving B and $F(i)$ can be given [BY94].

Predictions of the Model

We can now examine what these expressions tell us about the expected amount of work we need to do when doing hierarchical problem-solving: we examine $\mathsf{GoodTreeWork}(n)$ under various conditions. First it is useful to know the following results derived in the full report [BY94]:

$$\sum_{j=1}^{n-1} \mathsf{BadTreeWork}(j) = \begin{cases} O(b^k k^{n-1}) & p < 1/B \\ O(b^k k^{n-1} n) & p = 1/B \\ O(b^k k^{n-1} (pB)^{n-2}) & p > 1/B \end{cases} \tag{3}$$

1) In the region $0 \leq p \leq 1/B$, $\lim_{i \to \infty} F(i) = 1$. It can be shown that $\Gamma(i) \to (B-1)/2$ as $F(i) \to 1$.

Applying Eq. 3 we obtain:

$$\mathsf{GoodTreeWork}(n) = \begin{cases} O(b^k k^{n-1}) & pB < 1 \\ O(b^k k^{n-1} n) & pB = 1. \end{cases} \tag{4}$$

2) In the region $1/B < p < 1$, $\lim_{i \to \infty} F(i)$ lies between 1 and 0, decreasing as p increases. For any fixed value of p and B it can be shown that that $\Gamma(i)$ tends to a constant value independent of n, and that this value lies between 0 and $(B-1)/2$. Applying Eq. 3 we obtain:

$$\mathsf{GoodTreeWork}(n) \quad = \quad O(b^k k^{n-1} (pB)^{n-2}) \tag{5}$$

3) Finally, when $p = 1$ the DRP holds and $\forall i. F(i) = 0$. Hence, $\Gamma(i) = 0$ for all i and Eq. 3 simplifies to $k^{n-i} b^k \left(\frac{k^i - 1}{k-1}\right)$. Evaluating this expression at $i = n$ we obtain:

$$\mathsf{GoodTreeWork}(n) = b^k O(k^{n-1}). \tag{6}$$

Implications of the Analysis

There are two cases to consider: n constant and n variable. In certain domains we can make n, the number of abstraction levels, vary with ℓ. For example, in the Towers of Hanoi domain we can place each disk at a separate level of abstraction [Kno91]. In other domains, e.g., blocks world, it is not so easy to construct a variable number of abstraction levels, and n is generally fixed over different problem instances.

The length of the concrete-level solution is equal to k^n. Let $\ell = k^n$. We want to express our results in terms of ℓ. If we can vary n with ℓ then we can ensure that k remains constant and we have that $n = \log_k(\ell)$. In this case, b^k will become a constant. Otherwise, if n is constant, $k = \sqrt[n]{\ell}$ will grow slowly with ℓ. In this case, $b^k = b^{\sqrt[n]{\ell}}$ grows exponentially with ℓ, albeit much more slowly that b^ℓ (c.f., [Kno91]). This essential difference results in different asymptotic behavior for the two cases n variable and n constant. Table 1 gives the results of our analysis for these two cases expressed in terms of the length of solution ℓ.

Non-abstract search requires $O(b^\ell)$; hence, it is evident from the table that when $0 \leq p \leq 1/B$ and when $p = 1$ abstraction has a significant benefit. If we can vary n we can obtain an exponential speed-up, and even if n is not variable, we still obtain a significant speed up by reducing the exponent ℓ to its n-th root. Our result for $p = 1$ agrees with that of Knoblock [Kno91]: here we have the DRP and all of his assumptions hold. Our results for the region $0 \leq p \leq 1/B$, however, extend his analysis, and indicate that abstraction is useful when the probability of refinement is very low. What is happening here is that although the number of bad subtrees that must be searched is large, it does not require much effort to search them: most paths die out after only a small number of levels.

As p approaches $1/B$ we see that the search complexity increases by a factor of n, and as we move to the region $1/B < p < 1$ things are worse: we increase by a factor, $(pB)^n$, that is exponential in n. In these regions it is not always advantageous to increase the number of abstraction levels n, especially in the region $1/B < p < 1$. As p increases in the region $1/B < p < 1$ search first becomes harder and then becomes easier, as the number of bad subtrees to be searched drops

p	$[0, 1/B)$	$1/B$	$(1/B, 1)$	1
Variable n	$O(l)$	$O(l \log_k(l))$	$O(l(pB)^{\log_k(l)})$	$O(l)$
Constant n	$O(lb^{\sqrt[n]{l}})$	$O(lb^{\sqrt[n]{l}}n)$	$O(lb^{\sqrt[n]{l}}(pB)^n)$	$O(lb^{\sqrt[n]{l}})$

Table 1: Search Complexity for Different Regions of Refinement Probability.

off. Search complexity varies continuously until it again achieves the low complexity of $p = 1$ where the DRP holds.

Our analysis also tells us that if the number of possible refinements for an abstract solution (B) is large, then searching the abstraction tree is more expensive in the worst region $1/B < p < 1$. This is to be expected: the abstraction tree is bushier and in this region we have to search a significant proportion of it. Also of interest is that B does not play much of a role outside of this region, except, of course, that it determines the size of the region. Hence, if we know that the DRP holds,[6] or if the probability of refinement is very low, we do not have to worry much about the shape of the abstraction tree. However, without such assurances it is advantageous to choose abstraction hierarchies where abstract solutions generate fewer refinements. For example, this might determine the choice of one criticality ordering over an alternate one in ABSTRIPS-style abstraction.

A question that remains is how does one determine the refinement probability p? One method is to use a learning algorithm to keep track of the statistics of successful and unsuccessful refinements. Such statistics can be used to estimate p. Once such estimates are obtained they can be used to measure the merit of a particular abstraction hierarchy. It then becomes possible to construct an adaptive planner that can use these measurements to decide whether or not to use abstraction, to decide between alternate abstraction hierarchies, or even to automatically construct good abstraction hierarchies. We have implemented statistics gathering in a working planning system, and are currently investigating the design of an adaptive planner. We have also recently completed a series of experiments to provide empirical confirmation of the results presented here. These developments will be reported on in the full report [BY94].

References

[AN72] K. B. Athreya and P. E. Ney. *Branching Processes.* Springer-Verlag, New York, 1972.

[BY91] Fahiem Bacchus and Qiang Yang. The downward refinement property. In *Procceedings of the International Joint Conference on Artifical Intelligence (IJCAI)*, pages 286–292, 1991.

[BY94] Fahiem Bacchus and Qiang Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 1994. (60 pages in manuscript, to appear).

[CKT91] Peter Cheeseman, Bob Kanefsky, and Willian M. Taylor. Where the really hard problems are. In *Procceedings of the International Joint Conference on Artifical Intelligence (IJCAI)*, pages 331–337, 1991.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications: Volume 1.* John Wiley and Sons, New York, 1968.

[Kno91] Craig Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the AAAI National Conference*, pages 686–691, 1991.

[Kor85] Richard Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1985.

[KP83] R. M. Karp and J. Pearl. Searching for an optimal oath in a tree with random costs. *Artificial Intelligence*, 21:99–116, 1983.

[KTY91] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the AAAI National Conference*, pages 692–697, Anaheim, CA., 1991.

[Min63] Marvin Minsky. Steps towards artificial intelligence. In Edward A. Feigenbaum, editor, *Computers and Thought*, pages 406–450. McGraw-Hill, New York, 1963.

[MP91] C. J. H. McDiarmid and G. M. A. Provan. An expected-cost analysis of backtracking and non-bactracking algorithms. In *Procceedings*

[6] Various tests for detecting if the DRP holds of an abstraction hierarchy are given in [BY91].

of the International Joint Conference on Artifical Intelligence (IJCAI), pages 172–177, 1991.

[NS72] Allen Newell and A. Simon, Herbert. *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, N.J., 1972.

[NSS62] Allen Newell, J. C. Shaw, and Herbert A. Simon. The processes of creative thinking. In *COmtemporary Approaches to Creative Thinking*, pages 63–119. Altherton Press, New York, 1962.

[Sac74] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Sac77] Earl Sacerdoti. *A Structure for Plans and Behavior.* Elsevier, Amsterdam, 1977.

[Ste81] Mark Stefik. Planning with constraints. *Artificial Intelligence*, 16:111–140, 1981.

[Tat77] Austin Tate. Generating project networks. In *Procceedings of the International Joint Conference on Artifical Intelligence (IJCAI)*, pages 888–893, 1977.

[Ten89] Josh Tenenberg. Inheritance in automated planning. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the First Conference on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann, San Mateo, California, 1989.

[Wil84] David Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[YT90] Qiang Yang and Josh D. Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings of the AAAI National Conference*, pages 204–209, 1990.