Anytime Heuristic Search: First Results

Eric A. Hansen   Shlomo Zilberstein   Victor A. Danilchenko

CMPSCI Technical Report 97-50
September, 1997

Anytime Heuristic Search: First Results

Eric A. Hansen **Shlomo Zilberstein** **Victor A. Danilchenko**
Computer Science Department
University of Massachusetts
Amherst, MA 01003 U.S.A.
hansen,shlomo,danilche@cs.umass.edu

Abstract

We describe a simple technique for converting heuristic search algorithms into anytime algorithms that offer a tradeoff between search time and solution quality. The technique is related to work on use of non-admissible evaluation functions that make it possible to find good, but possibly sub-optimal, solutions more quickly than it takes to find an optimal solution. Instead of stopping the search after the first solution is found, however, we continue the search in order to find a sequence of improved solutions that eventually converges to an optimal solution. The performance of anytime heuristic search depends on the non-admissible evaluation function that guides the search. We discuss how to design a search heuristic that "optimizes" the rate at which the currently available solution improves.

1 Introduction

One of the most widely-used frameworks for problem-solving in artificial intelligence is heuristic search for a least-cost solution path through a tree or graph. A problem is specified by giving a start node, a goal node (or set of goal nodes), operators for moving from one node to the next, and costs for the operators. A solution path represents a sequence of steps for solving the problem represented in this way. There are well-known search algorithms, among them A* and AO*, for finding least-cost solution paths through trees and graphs of various kinds.

For large and complex problems, finding an optimal solution path may take a long time and a sub-optimal solution that can be found quickly may be more useful. Various techniques for modifying heuristic search algorithms to allow a tradeoff between solution quality and search time have been studied. All make the search non-admissible, either by using a non-admissible heuristic to start with, or by weighting an admissible evaluation function to make it non-admissible, e.g.,[Pohl, 1970; Harris, 1974; Ghallab & Allard, 1983; Pearl, 1984; Bagchi & Srimani, 1985; Davis et al., 1988; Chakrabarti et al., 1988; Koll & Kaindl, 1992]. In the substantial literature on these techniques, the assumption is virtually always made that the search stops as soon as the first solution is found. Analysis has focused on characterizing the tradeoff between the time it takes to find the first solution and its quality. Proving that this technique is ϵ-admissible, for example, involves proving that the first solution found is guaranteed to be within a factor ϵ of optimal [Pohl, 1973; Pearl & Kim, 1982; Ghallab & Allard, 1983; Davis et al., 1988; Koll & Kaindl, 1992].

In this paper, we begin with the simple observation that there is no reason not to continue a non-admissible search after the first solution is found. By continuing the search, a sequence of improved solutions can be found that eventually converges to an optimal solution. This observation has been made before in passing [Harris, 1974; Korf, 1993], but here we study it at length. We are intrigued by the fact that it provides a general technique for converting heuristic search algorithms into anytime algorithms. Anytime algorithms are useful for problem-solving under varying and uncertain time constraints because they have a solution ready whenever they are stopped, and the quality of the solution improves with additional computation time [Dean & Boddy, 1988; Horvitz, 1988]. Because heuristic search is used so widely, a general method for transforming heuristic search algorithms into anytime algorithms could prove useful for applications for which good anytime algorithms are not otherwise available.

In section 2, we describe how to transform any heuristic search algorithm that uses open and closed lists into an anytime algorithm. A similar strategy can be used to create anytime versions of search algorithms that use other methods of organizing the search. Our chief interest is not in describing how to do this – it is straightforward – but in studying the performance of the anytime heuristic search algorithm that results. The performance of an anytime algorithm can be characterized by a performance profile that predicts expected solution quality as a function of running time. (See figures 1 through 3.) Because different search strategies give rise to different performance profiles, in section 3 we discuss the difficult problem of how to "optimize" (so far as this is possible) the performance profile of anytime heuristic search, that is, how to conduct the search in such a way that the best possible anytime search algorithm results. Section 4 reports initial experiments that test the feasibility of our approach, and section 5 discusses some issues that we are continuing to investigate.

2 Anytime A*

We first describe how to convert a heuristic search algorithm that uses open and closed lists into an anytime algorithm. We use A* as an example, although the framework we describe can be applied to related memory-limited search algorithms [Chakrabarti et al., 1989; Russell, 1992]. All of these algorithms systematically search a space of possible solutions by maintaining two lists: an open list that contains nodes on the frontier of the search that are candidates for expansion, and a closed list that contains nodes that have already been expanded. (A closed list is necessary for graph search problems only, not for tree search problems.) Open nodes are selected for expansion in best-first order, based on an evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the least cost path currently known from the start node to n, and $h(n)$ is a heuristic estimate of $h^*(n)$, the cost of a minimum cost path from n to a goal node. If $h(n)$ is admissible, that is, if it never overestimates $h^*(n)$, then the first solution path found by A* is guaranteed to be optimal.

To convert A* to an anytime algorithm, we make two simple changes. First, we use a non-admissible evaluation function to select which node to expand next. We have wide latitude in choosing an evaluation function, and different evaluation functions will give rise to different performance profiles. To create a good anytime algorithm, we would like to optimize the rate at which the quality of the currently available solution improves. We discuss how to define such an evaluation function in the next section. For now, we simply note that nodes are selected for expansion based on an evaluation function that is non-admissible because we want to find a good, but not necessarily optimal, solution as quickly as possible.

A second change we make to A* is, of course, to continue the search after the first solution is found. Because the search is continued, an auxiliary, admissible evaluation function is still used. It provides a lower bound on the cost of the best solution path through a node and is used to prune the open list. Because the best solution found so far is an upper bound on the cost of an optimal solution, any node on the open list that has a lower bound – given by an admissible evaluation function – that equals or exceeds the current upper bound can be pruned. Pruning the open list is important because it makes it possible to detect convergence to an optimal solution. As soon as the open list is empty, the search algorithm has converged.

Weighted evaluation function

Our anytime version of A* uses a non-admissible evaluation function to select nodes for expansion and an admissible evaluation function to prune the open list. Let f denote the admissible evaluation function and let f' denote the non-admissible evaluation function. It is possible (although not necessary) for both evaluation functions to use the same heuristic h. We now briefly review a widely used method for using an admissible heuristic to create a non-admissible evaluation function that can find an approximate solution faster than it would take to find an optimal solution. Although this technique is not the only way to create a non-admissible evaluation function, it works well and provides a reference point to which we can compare other approaches.

Beginning with [Pohl, 1970], various researchers have explored the effects of weighting the two factors $g(n)$ and $h(n)$ in the node evaluation function of heuristic search differently. In general, $f(n) = (1 - w) * g(n) + w * h(n)$, where the weight w is a parameter set by the user. (Or equivalently, $f(n) = g(n) + w'h(n)$, where $w = \frac{w'}{1+w'}$.) If $w \leq 0.5$, the resulting search is admissible as long as the h-heuristic is admissible. But if $w > 0.5$, the (first) solution found may not be optimal, although it is often found much faster because adding a distance-dependent weight to h gives the search more of a depth-first aspect. An appropriate setting of w makes possible a tradeoff between the quality of the solution found and computation time.

A weighted evaluation function can be used with any heuristic search algorithm, and not just those that use open and closed lists. For example, Korf (1993) uses this technique with RBFS and Chakrabarti *et al.* (1988) show how to use it with AO*. These heuristic search algorithms and others can also be continued after the first solution is found, creating anytime algorithms. The details of how to transform them into anytime algorithms vary depending on how each keeps track of its progress through the search space, but the differences from what we have described for A* are minor.

Use of this technique raises the question: what weight provides the best performance? A weight of 0.6 creates one anytime algorithm and a weight of 0.75 creates another. In some cases, it is possible to improve search performance by adjusting the weight dynamically with the depth or progress of the search [Pohl, 1973; Koll & Kaindl, 1992]. Is there a principled way of developing a good heuristic evaluation function for anytime search aside from simple trial-and-error testing of different weights to find the best one for a given problem? Is a weighted evaluation function even the best way to design a non-admissible evaluation function for anytime search? In the rest of this paper we address these questions.

3 Optimizing search effort

Because admissable evaluation functions do not consider search effort, or the potential tradeoff between search effort and solution quality, they are unsuitable for resource-bounded search. Weighting the h-cost component of an evaluation function more heavily to make it non-admissable can accelerate search for a solution because it makes nodes that are closer to a solution seem more attractive. In general, the lower the h-cost, the less search effort is needed to complete a solution from a node and the more attractive that node should be from the point of view of finding a good (or improved) solution quickly. In this way, a weighted evaluation function has the effect of implicitly adjusting a tradeoff between search effort and solution quality. What we would like to do is make this tradeoff between search effort and solution quality explicit in the heuristic evaluation function so that we can optimize search effort directly, rather than relying and trial-and-error to design an evalution function that results in good anytime performance for a particular problem.

What does it mean to optimize search effort? For anytime algorithms that return a stream of improving solutions, we take it to mean optimizing the rate at which solution quality improves as a function of search time. In other words, an anytime algorithm should try to improve the currently available solution as fast, and by as much, as possible. Selecting nodes for expansion in an order that minimizes the following evaluation function "optimizes" search performance in this sense:

$$f'(n) = \frac{\text{expected search effort}}{\text{expected improvement in solution quality}}$$

There may be more than one way to estimate this ratio. One possibility is to define expected search effort as,

$$\sum_{h'(n)} Pr(h'(n))SE(h'(n)),$$

where $h'(n)$ denotes the length of the next solution path found from node n to a goal node, $Pr(h'(n))$ denotes the probability that the next solution path found from node n to the goal will have length $h'(n)$, and $SE(h'(n))$ denotes the search effort (in time or nodes expanded) for finding this path. We can then define expected improvement in solution quality as,

$$\sum_{h'(n)<l-g(n)} Pr(h'(n))(l - (g(n) + h'(n))),$$

where l denotes the length of the current best solution. Before the first solution is found, l represents the penalty for not finding a solution. (Because l will change in the course of the search, the heuristic value of a node can change after it is opened and the open list may need to be resorted in the course of the search. Although there are possible ways of overcoming this inefficiency, we do not discuss them here because our initial interest is to appraise how well this heuristic evalution function works.)

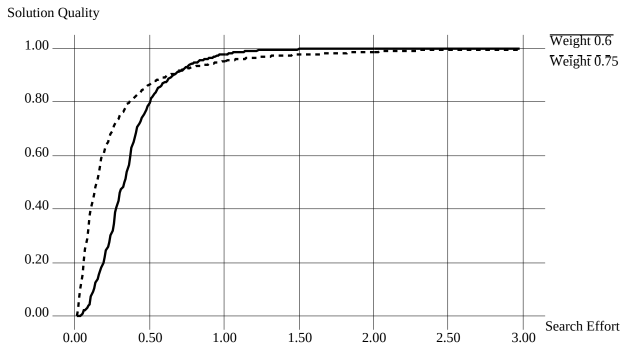

Figure 1: Performance profiles of anytime search with two different weights.

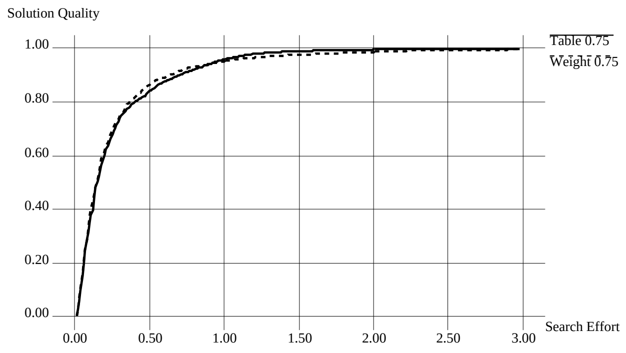

Figure 2: Performance profiles with weight=0.75 and the corresponding evaluation using statistical estimates.

Our technique for designing an evaluation function that optimizes search effort relies on statistics, Pr and SE, that characterize the performance of anytime heuristic search using this evalution function. But these statistical estimates are not available before the algorithm that uses them has been run. To avoid this circularity, we initially use statistics gathered by running anytime heuristic search with a weighted evalution function. Then by running the algorithm with our evalution function, using these statistical estimates, we can generate new estimates that more accurately reflect the behavior of anytime heuristic search with the evalution function we have defined. This suggests a method of iteratively improving an evalution function. As we generate more accurate estimates of search performance, we can expect search performance using an evalution function that relies on the accuracy of these estimates to improve. This process could continue until search performance "converges."

4 Example and Analysis

We first tested the performance of anytime A* on the 8-puzzle using a weighted evaluation function. Figure 1 shows the performance profile of anytime A* using weights of 0.75 and 0.6. Solution quality is normalized with respect to the optimal solution. Search ef-

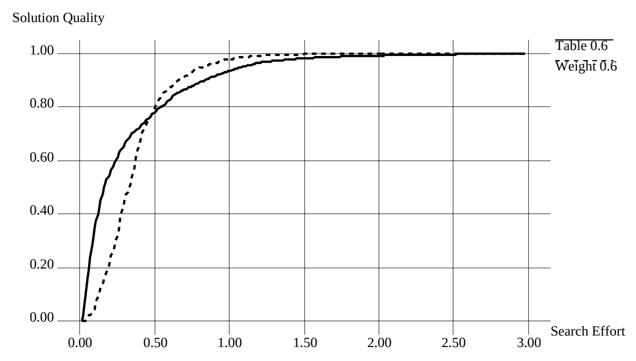

Figure 3: Performance profiles with weight=0.6 and the corresponding evaluation using statistical estimates.

fort is normalized with respect to the number of nodes expanded by standard A*; that is, for each problem instance, search effort is measured as a fraction of the number of nodes A* expands to find an optimal solution to the same problem. To average results for randomly generated 8-puzzle problems we also weighted problems according to search effort, reflecting the fact that it is more important to perform well on hard problems. Note that although anytime A* usually expands more nodes than A* before converging, it often finds (what turns out to be) an optimal solution before standard A*. Note also that weighting h more heavily tends to create more of an anytime effect, as would be expected.

We used anytime A* with a weighted evaluation function to estimate both probable improvement of solution quality and search effort, gathering statistics from one thousand randomly generated 8-puzzle problems to estimate $Pr(h'(n)|h(n))$ and $SE(h'(n)|h(n))$, conditioned on the Manhattan distance heuristic $h(n)$. Figure 2 shows the performance profile of anytime heuristic search when its evaluation function uses statistical estimates based on a weighted evaluation function with a weight of 0.75. Its performance profile is almost the same as when the weighted evaluation function is used. Figure 3 shows the performance profile of anytime heuristic search using when its evaluation function uses statistical estimates based on a weighted evaluation function with a weight of 0.6. Here we notice a performance difference. Initially, anytime search using the statistics-based evaluation function performs better. But eventually it performs somewhat worse. It is noteworthy that the performance profiles of the two statistics-based algorithms are almost the same, regardless of the weighted version used to collect statistics. This suggests that the statistics-based algorithm is not sensitive to initial estimates of the probabilities and that updating the statistics tends to cause the algorithm to converge to a particular performance profile. More work is needed to assess the convergence of this process.

The statistics-based approach to designing anytime search algorithms produced good anytime behavior, confirming our initial intuition. However, the results of our initial evaluation provide little or no reason to prefer a statistically-based evaluation function that tries to optimize search effort directly to a simple weighted evaluation function with a weight that can be adjusted by trial-and-error to fine-turn performance. Although more work is needed to assess this, we are encouraged to have observed that the anytime approach in general tends to perform better for hard problems for which A* must expand many nodes to find an optimal solution. In the rest of this section, we make a few other observations that emerged from our study.

Despite its apparent simplicity, the 8-puzzle may be a particularly challenging problem for an anytime approach. Solutions to the 8-puzzle problem are sparse and at uneven and unpredictable depths in a search tree, making it possible to search long paths of the search tree without finding a solution. We also observed some apparent anomalies when the Manhattan distance heuristic is used to estimate search effort. For example, the statistics we gathered revealed that puzzle-states for which the value of the Manhattan distance heuristic is even tend to be somewhat harder to solve, on average, than puzzle-states for which the value of the Manhattan distance heuristic is odd. This tends to create a very slight tendency for the statistics-based heuristic to avoid even-numbered puzzle-states, a tendency that is obviously self-defeating.

The evaluation function we constructed can only be expected to perform consistently if $h(n)$, the Manhattan distance heuristic, is an unbiased predictor of probably solution improvement and search effort at every point in the search tree. One possible explanation of some of the results we obtained is that this assumption holds for nodes that are shallow in the search tree, but that the sampling of nodes deep in the search tree is not random and depends on the path taken to reach these nodes. This may lead us to a different design of the statistical estimates in the future.

Finally, we note that it may be unrealistic to expect that one approach to designing anytime algorithms will totally dominate all other approaches – for any possible time duration. Instead, one anytime algorithm may provide better short-term performance whereas another anytime algorithm provides better long-term performance. In that case, a node evaluation function that optimizes search effort will also need to take into account the probable stopping time of the algorithm. If the deadline is imminent, for example, it may be advantageous to emphasize the depth-first aspect of the search more than it otherwise would be.

The heuristic evaluation function we defined earlier maximizes the rate of improvement by considering only the next solution that is found. In other words, it is greedy. While this seems reasonable, in some cases it may be more advantageous to consider likely improvement over a time interval t during which many solutions can be found. To test this hypothesis, we experimented with the following heuristic:

$$f'(n) = \sum_{h'(n) < l - g(n)} Pr^t(h'(n))(l - (g(n) + h'(n))$$

In this case, Pr^t denotes the probability of a solution of length $h'(n)$ after time t. These probabilities vary with the "horizon" of the search, meaning we must gather statistical estimates for each of a number of horizons. Our experiments with this heuristic were successful to this extent: using a shorter horizon increased the anytime behavior of the search. The longer the horizon, the more the search acts like standard A*. However, further evaluation of this approach is needed in order to assess its characteristics.

5 Related Work

Previous work on resource-bounded heuristic search has adopted the model of Korfs's real-time A* algorithm (RTA*) which assumes that search is interleaved with execution [Korf, 1990]. After searching for some bounded amount of time, the best next action is chosen and the search-act cycle repeats until the goal state is reached. Examples include DTA* [Russell & Wefald, 1991], BPS [Hansson & Mayer, 1990], DYNORA [Hamidzadeh & Shekhar, 1991], and k-best [Pemberton, 1995]. Because real-time search algorithms commit to actions before finding a complete solution, they cannot (typically) find optimal solutions. In contrast, we assume that a search phase precedes an execution phase and that the output of the search is a complete solution. Whereas real-time search algorithms try to find the *best next decision* under a time constraint, our anytime approach tries to find the *best complete solution* under a time constraint.

We know of no previous work on meta-level control of heuristic search for complete solutions. However, the idea of searching for a complete solution that is suboptimal and then continuing to search for improved solutions is not new. Branch and bound strategies often have this effect. In fact, the strategy we have employed can be regarded as best-first branch-and-bound search using a non-admissible heuristic for branching and an admissible heuristic for bounding (pruning).

A technique similar to the one we have exploited to create an anytime version of A* has been used to similar effect for IDA*, although for a different reason. IDA* performs poorly on problems such as the traveling salesman where nodes have distinct f-costs because it may expand only one new node each iteration. To prevent excessive iterations and node re-generations, algorithms such as IDA*-CR [Sarkar et al., 1991] and DFS* [Rao, Kumar, & Korf, 1991] set successive thresholds higher than the minimum cost that exceeded the previous threshold using a technique similar to a weighted evaluation function. As a result, the first solution found is not guaranteed to be optimal. After finding an initial solution, these algorithms revert to depth-first branch-and-bound to ensure eventual convergence to an optimal solution. This approach to reducing node re-generations has the side-effect of creating an anytime algorithm in which an initial, sub-optimal solution is improved by continuing to search until an optimal solution is found. Korf (1993) notes that the same technique can be applied to RBFS, a linear-space best-first search algorithm.

We have focused on converting heuristic search algorithms into anytime algorithms that can be guaranteed to eventually converge to an optimal solution. Another approach to anytime search for complete solutions is local optimization. For example, [Ratner & Pohl, 1986] describe two local optimization algorithms for the n-puzzle. A drawback of this approach is that it generally converges to local optima. However, it may be advantageous to use local optimization to find initial solutions and then switch to anytime heuristic search to ensure eventual convergence to an optimal solution.

6 Conclusion

We have described a general technique for converting heuristic search algorithms into anytime algorithms that offer a tradeoff between search time and solution quality. We have also discussed how to create a heuristic evaluation function that optimizes the rate at which the quality of the currently available solution improves. Preliminary results for this work show that both techniques produce good anytime algorithms that can improve solution quality gradually as computation time increases. These algorithms can produce near optimal solutions with substantial savings of computation time. Thus far, the more disciplined, statistics-based approach does not appear to have the performance benefits one may expect over the weighted approach. But further evaluation of the approach in different domains is necessary in order to fully understand its characteristics.

Our anytime technique easily extends to memory-bounded versions of A* that "retract" nodes from the open list to keep its size manageable [Chakrabarti et al., 1989; Russell, 1992], and we plan to implement a memory-bounded version of anytime heuristic search that can be tested on problems with larger search spaces. We also plan to test this approach on more varied and realistic search problems. The range of problems for which an anytime version of heuristic search may be used is extensive, including scheduling, planning, and combinatorial optimization of many kinds. We believe this justifies further study of how to design a search heuristic that "optimizes" a tradeoff between solution quality and search effort.

References

[Bagchi & Srimani, 1985] Bagchi, A. and Srimani, P.K. 1985. Weighted heuristic search in networks. *Journal of Algorithms* 6:550-576.

[Chakrabarti et al., 1988] Chakrabarti, P.P.; Ghosh, S.; & DeSarkar, S.C. 1988. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence* 34:97-113.

[Chakrabarti et al., 1989] Chakrabarti, P.P; Ghosh, S.; Acharya, A.; & DeSarkar, S.C. 1989. Heuristic search in restricted memory. *Artificial Intelligence* 47:197-221.

[Davis et al., 1988] Davis, H.W.; Bramanti-Gregor, A.; & Wang, J. 1988. The advantages of using depth and breadth components in heuristic search. In *Methodologies for Intelligent Systems 3*, ed. by Z.W. Ras & L. Saitta, pp. 19-28.

[Dean & Boddy, 1988] Dean, T. and Boddy, M. 1988. An analysis of time-dependent planning. In *AAAI-88*, pp. 49-54.

[Ghallab & Allard, 1983] Ghallab, M. and Allard, D.G. 1983. A_ϵ – An Efficient near admissible heuristic search algorithm. In *IJCAI-83*, pp. 789-791.

[Hamidzadeh & Shekhar, 1991] Hamidzadeh, B. and Shekhar, S. 1991. DYNORA: A Real-time planning algorithm to meet response time constraints in dynamic environments. In *Proc. of IEEE Conf. on Tools for Artificial Intelligence*, pp. 228-235.

[Hansson & Mayer, 1990] Hansson, O. and Mayer, A. 1990. Probabilistic heuristic estimates. *Annals of Mathematics and Artificial Intelligence* 2:209-220.

[Harris, 1974] Harris, L.R. 1974. The heuristic search under conditions of error. *Artificial Intelligence* 5:217-234.

[Horvitz, 1988] Horvitz, E. 1988. Reasoning under varying and uncertain resource constraints. In *AAAI-88*, pp. 111-116.

[Koll & Kaindl, 1992] Koll, A.L. and Kaindl, H. 1992 A new approach to dynamic weighting. In *ECAI-92*, pp. 16-17.

[Korf, 1990] Korf, R.E. 1990. Real-time heuristic search. *Artificial Intelligence* 42:197-221.

[Korf, 1993] Korf, R.E. 1993. Linear-space best-first search. *Artificial Intelligence* 62:41-78.

[Pemberton, 1995] Pemberton, J.C. 1995. k-best: A new method for real-time decision making. In *IJCAI-95*, pp. 227-233.

[Pearl, 1984] Pearl, J. 1984. *Heuristics: Intelligent search strategies for computer problem solving.* Addison-Wesley.

[Pearl & Kim, 1982] Pearl, J. and Kim, J.H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4(4):392-399.

[Pohl, 1970] Pohl,I. 1970. First results on the effect of error in heuristic search. *Machine Intelligence* 5:219-236.

[Pohl, 1973] Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI-73*, pp. 12-17.

[Rao, Kumar, & Korf, 1991] Rao, V.N.; Kumar, V.; & Korf, R.E. 1991. Depth-first vs. best-first search. In *AAAI-91*, pp. 434-440.

[Ratner & Pohl, 1986] Ratner, D. and Pohl, I. 1986. Joint and LPA*: Combination of approximation and search. In *AAAI-86*, pp. 173-177.

[Russell, 1992] Russell, S. 1992. Efficient memory-bounded search methods. In *ECAI-92*, pp. 1-5.

[Russell & Wefald, 1991] Russell, A. and Wefald, E. 1991. *Do the Right Thing: Studies in Limited Rationality* MIT Press.

[Sarkar et al., 1991] Sarkar, U.K; Chakrabarti, P.P.; Ghose, S; & DeSarkar, S.C. 1991. Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence* 50(2):207-221.