ITSA*: Iterative Tunneling Search with A*

David A. Furcy
University of Wisconsin
Computer Science Department
800 Algoma Boulevard
Oshkosh, WI 54901-8643
furcyd@uwosh.edu

Abstract

This paper describes a new approach to anytime heuristic search based on local search in the space of solution paths. This work makes two contributions. First, we introduce a new local optimization algorithm called Iterative Tunneling Search with A* (ITSA*) that explores the neighborhood of a given solution path in order to find shortcuts and thus a shorter overall path. Second, we introduce a new anytime heuristic search algorithm called ABULB that performs a full-fledged local (or neighborhood) search with restarts. ABULB uses a variant of beam search called BULB to generate an initial path and ITSA* to locally optimize this path. When a minimum is reached, ABULB restarts BULB to obtain the next path to optimize, etc. The successive paths output by BULB and ABULB have smaller and smaller costs. We present empirical results with this new anytime algorithm in two standard benchmark domains.

Introduction

Local or neighborhood search is a popular approach for solving combinatorial optimization problems in both the Operations Research (OR) and Artificial Intelligence (AI) communities (Russell & Norvig 1995; Aarts & Lenstra 1997; Papadimitriou & Steiglitz 1998; Voss *et al.* 1999; Ribeiro & Hansen 2001). However, local search methods are typically not applied to the shortest-path problem in a graph. One reason is that, unlike in typical optimization problems such as the Traveling Salesperson Problem or TSP (Lawler *et al.* 1985), finding any solution (even a high-cost solution) to a shortest-path problem may be expensive since it requires search. Second, the solution space contains structured elements namely paths, not states, like in standard heuristic search. As a result, it is not trivial to define a good neighborhood structure on paths (namely one that induces a surface with few local optima) that can be searched efficiently. Third, since building a starting solution (any solution) to the shortest-path problem is computationally expensive, it is an open issue how to efficiently identify additional restarting solutions to get the search agent out of local optima.

In this paper, we describe a new anytime heuristic search algorithm called ABULB (for Anytime BULB). It is a local search algorithm[1] in the space of solution paths that addresses the three aforementioned issues. In (Furcy 2004), we introduced BULB, a variant of beam search with backtracking and showed that it efficiently finds sub-optimal solutions to large heuristic search problems. First, ABULB uses BULB to generate an initial solution.

Second, we define a neighborhood structure on the set of solution paths. The neighborhood of a path is limited by the amount of available memory and contains the set of nodes that are close to the path found by BULB, where the closeness of two paths is defined in terms of the smallest number of edges connecting the states on one path to the states on the other path. This neighborhood is built up and searched by an algorithm called Iterative-Tunneling Search with A* (or ITSA*, pronounced *It's a star!*). In the local optimization loop, ITSA* is repeatedly executed to find better and better paths.

Third, when a local optimum is reached, ABULB restarts BULB in order to find a new solution path to start from. This path is then locally optimized with ITSA*, and the process repeats until time runs out.

The two main contributions of this work are the local optimization algorithm called ITSA* and the anytime extensions of the BULB algorithm. We also combined both approaches into a local search algorithm with restarts called ABULB. Both ITSA* and ABULB are anytime algorithms, which means that they find a first, possibly sub-optimal path quickly and then keep searching for and output solution paths with lower and lower costs until time runs out (Dean & Boddy 1988; Boddy & Dean 1989; Boddy 1991).

In the rest of the paper, we first give an overview of the BULB algorithm. Then, we introduce the idea of iterative tunneling and describe its implementation in the ITSA* algorithm. In the following section, we report on empirical results with two variants of ITSA*. Next, we discuss sev-

[1]Note that local or neighborhood search algorithms are sometimes called meta-heuristics, e.g., (Voss *et al.* 1999; Ribeiro & Hansen 2001), where the word *heuristics* refers to approximation algorithms. In contrast, heuristic search algorithms like A* (Hart, Nilsson, & Raphael 1968) find optimal solutions using heuristic functions.

eral ways of transforming BULB into an anytime heuristic search algorithm, namely by 1) simply letting it run after it finds a solution path, 2) restarting it with a different parameter setting, or 3) interleaving its execution with ITSA*. Finally, we discuss related and future work, and conclude.

The BULB Algorithm

An initial solution on which to perform local optimization is needed. In the shortest-path problem, we must expend some effort searching for an initial solution, as opposed to say, in the TSP, where any permutation of the cities is a valid initial solution. To quickly find an initial solution path, we may use any one of a number of approximation algorithms. The most well-known approximation heuristic search algorithms are Weighted A* or WA* (Pohl 1970; 1973; Gaschnig 1979; Pearl 1985) and beam search (Bisiani 1987; Winston 1992; Zhou & Hansen 2004). In (Furcy 2004), we showed that beam search scales up to larger problems and finds better solution paths than improved variants of WA* described in (Furcy & Koenig 2005b). We thus choose beam search. In (Furcy & Koenig 2005a), we transformed beam search into a complete algorithm called BULB and showed that it scales up to larger problems than beam search. In this work, we use BULB to generate the initial solution path to be optimized.

BULB makes beam search complete by adding backtracking. One way to define beam search is to modify breadth-first search so that it only keeps at most a constant number B of nodes at each level of its search tree. B is called the *beam width*. One iteration of beam search expands all the nodes at the current level. The set of B nodes with the smallest heuristic values among the successors make up the beam at the next level down in the search tree. Since there are at most B nodes at each level in the tree, the memory consumption of beam search is linear in the search depth. But because it prunes all but the best B nodes at each level, beam search may prune all the paths to the goal. When this happens, beam search will terminate without finding a solution path because either the next level of the tree becomes empty (all the successors are already in memory) or the memory is full. In either case, misleading heuristic values led beam search astray. To remedy this problem, BULB adds backtracking to beam search. Since heuristic values are typically less informed at the top of the tree (which is far from a goal state) and chronological backtracking (like in depth-first search) takes a long time to backtrack all the way to the top, BULB uses a different backtracking strategy based on limited-discrepancy search (Harvey & Ginsberg 1995), which backtracks to the top of the tree much faster than depth-first search. In short, BULB generalizes beam search to beam search with backtracking and limited discrepancy search to beam widths larger than one. For additional details on BULB, the reader is referred to (Furcy & Koenig 2005a).

Iterative Tunneling

In this section, we first motivate our use of local optimization. Then we describe the idea of iterative tunneling and the associated neighborhood structure it imposes on the space of solution paths. Finally, we describe one implementation of iterative tunneling which we call ITSA*.

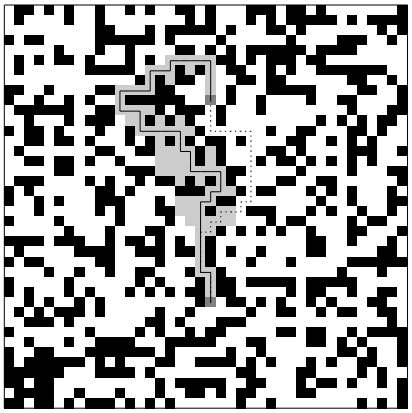

Figure 1: Solutions found (unbroken line) and missed (dashed line) by WA* with $f = g + 5 \times h$ in a grid-world problem.

Motivation

Approximation algorithms sacrifice solution quality for reduced memory consumption and run-time. For example, WA* and beam search (and thus BULB as well) rely heavily on the heuristic function since they delay the expansion of nodes with the highest heuristic values (in the case of WA*) or even prune them altogether (in the case of beam search and BULB). All these approximation algorithms thus have a strong greedy flavor. While greedy search with perfect information leads straight to the goal, it likely leads the search astray when combined with imperfect heuristic values. Nevertheless, WA*, beam search, and BULB are often able to eventually reach the goal. But these two characteristics of approximation algorithms together imply that the solution path they find is often convoluted.

It is therefore possible that approximation algorithms overlook shorter solutions that are actually close (in the space of solution paths) to the area of the search space that they have explored. This possibility is confirmed empirically. For example, Figure 1 depicts a concrete example of this situation for WA* with the Manhattan distance heuristic in a grid-world domain, whose two-dimensional structure enables a direct visualization of its search frontier. Obstacles are black, expanded states are light gray, and the start and goal states are dark gray (with the goal on top). The figure illustrates how the convoluted path found is longer (fifty steps) than but close to the optimal path (thirty steps). Such patterns are also common for beam search and BULB as our empirical evaluation of local optimization demonstrates (see below).

Local Optimization with Iterative Tunneling

This observed behavior of approximation algorithms led us to consider the following approach. In order to avoid leaving some interspersed regions of the state space unexplored, we propose to search systematically the neighborhood of a given solution path P, find a minimum-cost path P' in this

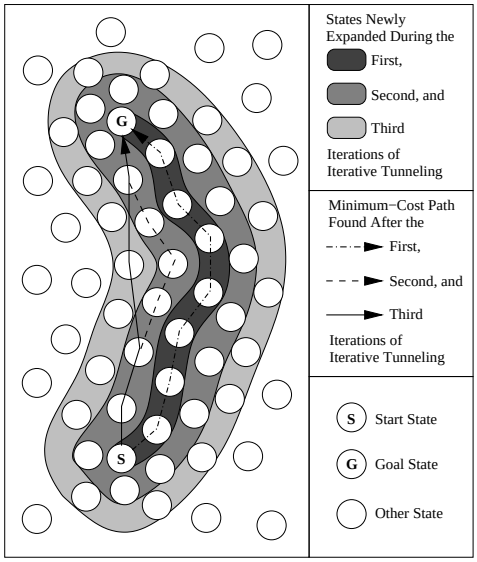

Figure 2: Iterative tunneling defines the neighborhood of a path.

neighborhood, then compute and search the neighborhood centered around P', and repeat the process until a local minimum is found. The size of each neighborhood region is equal to the maximum number of nodes that fit in memory. Each neighborhood is built incrementally by including all nodes that can be reached in one step from any node already in the neighborhood, starting with the initial neighborhood containing the set of nodes on the current solution path P. In other words, the neighborhood of a path P may be built using a breadth-first search seeded with all the states on P. The analogy we use to describe this process is that of *iterative tunneling*: If the solution path from start S to goal G is seen as a narrow tunnel dug through the search space, then each iteration enlarges the diameter of the tunnel until there is no available space to heap any additional evacuated debris, assuming memory is viewed as a container for debris/nodes (see Figure 2).

ITSA*

Given the neighborhood of a solution path P, we want to locally optimize P by finding the shortest path from start to goal entirely contained in its neighborhood. We could first identify the set of states that make up the neighborhood with a breadth-first search seeded with all the states on P and that terminates when memory is full, and then search this neighborhood with A*. However, this approach has two drawbacks. First, it requires two full explorations of the neighborhood, one with breadth-first search and one with A*. Second, because A* delays the expansion of states with large heuristic values, some states in the neighborhood may never be visited by A* and therefore need not be stored in the neighborhood at all. Unfortunately, these states cannot be identified during the initial breadth-first search.

To address both problems, we propose to interleave the building of the neighborhood with its search in order to take advantage of the pruning power of A* during construction and thus have the neighborhood contain as many promising states as possible. We call the resulting algorithm ITSA* for Iterative Tunneling Search with A* (pronounced *It's a star!*). ITSA* iteratively performs modified A* searches from the start to the goal. ITSA* assigns each generated state s an iteration number. This number corresponds to the distance from the initial path P to s computed as the smallest number of edges on a path from any state on P to s. Once assigned to s when it is first generated, this number never changes. Before the call to ITSA*, all states on P (and their other successors, respectively) are stored in memory with an iteration number equal to zero (one, respectively). Then A* is run repeatedly with an increasing iteration number (starting at one) until memory runs out. At the beginning of each iteration, the OPEN list is initialized with the start state. A* proceeds as usual except that 1) it only inserts into the OPEN list states whose number is less than or equal to the iteration number, and 2) each newly generated state is assigned a number equal to one more than the number of its parent in the search tree. Each iteration ends when A* is about to expand the goal state (except possibly for the last iteration, which ends when the memory is full).

Instead of performing a full exploration of a pre-existing neighborhood built with breadth-first search, ITSA* performs several A* searches over a region of the search space that grows around P in a layered fashion. Only the last iteration of ITSA* is a complete A* search over the entire neighborhood.

Empirical Evaluation of ITSA*

We evaluated the performance of ITSA* as a local optimization algorithm in two standard benchmark domains, namely the 48-Puzzle with the Manhattan distance heuristic and the Rubik's Cube with the PDB heuristic used in (Korf 1997). We ran ITSA* on solution paths found by BULB in 50 random instances for each domain[2]. We report the solution quality output by ITSA* and its run-time, which we compare to that of BULB (see Tables 1 through 4).

First, we discuss the performance of *one-step ITSA** (see Tables 1 & 2), where ITSA* is applied only once to the path found by BULB. In each domain, the absolute run-time of ITSA* remains approximately constant (always under ten seconds) when compared to the run-time of BULB. Since ITSA* searches a neighborhood whose size (i.e., its number of states) is fixed by the available memory, its run-time is essentially determined by the time it takes A* to search it. We speculate that the differences in run-times for various values of B result at least in part from the fact that ITSA* performs varying numbers of iterations of A* depending on the length (or equivalently the cost) of the starting solution path. This effect is more prominent in the Rubik's Cube domain, but in either case, the relation between initial solution length and run-time is not monotonic. Other factors influencing run-times include the overhead of node generation and the branching factor of the search space (both of which are larger in the Rubik's Cube than in the 48-Puzzle). The approximately constant run-time of one-step ITSA*, together

[2]Details of the empirical setup can be found in (Furcy 2004).

with the increasing run-time of BULB, explains why the relative increase in run-time of one-step ITSA* gets smaller as B increases (i.e., as the starting solution cost decreases).

Similarly, the relative improvement in solution cost achieved by ITSA* decreases as B increases. This trend is explained by the fact that, as the initial path length decreases, the path itself becomes less convoluted. Consequently, ITSA* has fewer opportunities to find shortcuts within the neighborhood. In other words, high-quality solutions are more likely to be (closer in solution quality to) local optima in the space of solution paths and, not surprisingly, ITSA* has a harder time improving on higher quality solutions (ceiling effect).[3]

Second, we discuss the performance of *multi-step ITSA**(see Tables 3 & 4), where we apply ITSA* iteratively, first to the path found by BULB, and then repeatedly to the best path found during the previous execution of ITSA*. Multi-step ITSA* can stop as soon as ITSA* returns its starting path which is then a local minimum in the space of paths. For efficiency reasons, instead of checking the path itself, we only check its length. So we stop multi-step ITSA* when the solution cost it returns is equal to that of its starting solution.

In both domains, the absolute run-time of multi-step ITSA* tends to decrease as B increases. As the initial path cost decreases, it is harder for local search to improve it and ITSA* is called fewer times before it reaches a local minimum. In addition, since the run-time of BULB increases with B, the relative increase in run-time of multi-step ITSA* decreases as B increases. Finally, both the absolute and relative improvements in solution quality decrease as B increases because again, better solutions are harder to improve on.

Local Search with Restarts

To get out of local minima, we need to find other solutions to restart from. We modified BULB so that it outputs a sequence of paths whose costs get smaller and smaller. We call the resulting algorithm Anytime BULB or ABULB. We investigated several anytime search schemes. The first version of ABULB simply runs BULB until the first path is found and then continues running with decreasing depth cutoffs since every time it finds a new solution path P, ABULB can prune all paths whose cost is larger than that of P. The second version of ABULB stops BULB every time it finds the first solution path with length say, L, and restarts it from scratch with a new, larger beam width equal to M/L, where M is the maximum number of nodes that can be stored in memory. If all edges have uniform costs, then the length of a path is the same as its cost and ABULB not only finds shorter and shorter paths, it is also guaranteed to eventually find the optimal path when given enough time. More details on these two variants of ABULB are available in (Furcy 2004) where we showed that the second variant of ABULB decreases solution cost faster than the first variant. We thus use ABULB version 2 from now on.

[3]The case $B = 10$ stands out in Table 2. With such a narrow beam width, BULB reaches deeply into the search space and the cost of the solution is over 100,000. Given that 1) the branching factor of the Rubik's Cube is approximately equal to 13 and 2) the available memory can store up to 3 million nodes, ITSA* can only perform one complete iteration of A* before memory runs out. The neighborhood only extends as far as layer one and the relative improvement in solution quality is thus low. Similarly, because BULB is so slow in this case, ITSA*'s relative increase in run-time is small.

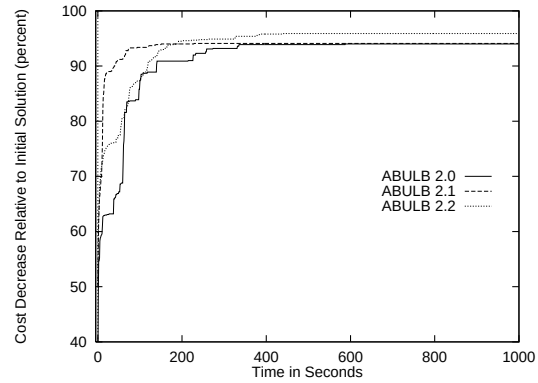

Figure 3: Combining ITSA* with ABULB version 2 in the 48-Puzzle (with 6 million nodes and B=5)

Up to this point, ABULB (version 2.0) does not use ITSA*. One obvious way to combine ABULB with ITSA* is to use ITSA* to locally optimize each solution produced by ABULB. Since ABULB 2.0 restarts BULB from scratch after each path is found, we can prune the current beam from memory (except for the path itself) and thus make full use of memory to store the neighborhood searched by ITSA*. We refer to the combination of ABULB 2.0 with one-step and multi-step ITSA* by the names ABULB 2.1 and 2.2, respectively. ABULB 2.0 refers to plain vanilla ABULB.

Figure 3 contains the performance profiles of all three versions of ABULB in the 48-puzzle. Both versions of ABULB using ITSA* outperform plain vanilla ABULB since both decrease solution cost faster than ABULB 2.0. ABULB 2.2 converges to lower-cost solutions than the other two versions. Furthermore, in about 7 minutes and using only the Manhattan distance heuristic, it finds solutions with an average cost of 481, which is little more than twice the estimated optimal cost. More detailed results and analyses are available in (Furcy 2004).

Related and Future Work

The closest algorithms to ITSA* are LPA* and Joint (Ratner & Pohl 1986), two local optimization algorithms that break down a path into segments and optimize each segment with a full A* search. Unlike ITSA*, these algorithms take two parameters and are not memory-bounded. One approach left for future work would use LPA* before ITSA* when BULB outputs such a long solution that ITSA* cannot shorten it, like when $B = 10$ in Table 2.

There exist only a few anytime heuristic search algorithms to date. The most relevant ones are two variants of WA*. First, Anytime A* (ATA*) (Hansen & Zilberstein 1996; Hansen, Zilberstein, & Danilchenko 1997) is a variant of WA* that does not stop after a solution is found. Therefore, ATA* is to WA* as ABULB 1 is to BULB. Sec-

B	BULB		BULB + one-step ITSA*					
	time (seconds)	cost	time (seconds)			cost		
			value	increase over BULB		value	decrease over BULB	
				absolute	relative		absolute	relative
5	0.1	11,737	5.7	5.6	5,600%	3,140	8,597	73%
10	0.9	36,282	6.7	5.8	644%	3,233	33,049	91%
100	6.1	14,354	12.2	6.1	100%	2,052	12,302	86%
1,000	7.3	1,409	12.8	5.5	75%	746	663	47%
10,000	21.7	440	27.7	6.0	28%	428	12	3%

Table 1: Performance of one-step ITSA* on paths found by BULB in the 48-Puzzle (with 6 million nodes in memory)

B	BULB		BULB + one-step ITSA*					
	time (seconds)	cost	time (seconds)			cost		
			value	increase over BULB		value	decrease over BULB	
				absolute	relative		absolute	relative
10	96.9	108,804.8	100.7	3.8	4%	94,346.6	14,458.2	13%
100	5.1	1,893.9	7.9	2.8	56%	679.0	1,214.9	64%
1,000	7.4	275.8	10.2	2.8	38%	178.5	97.3	35%
10,000	13.8	53.6	18.5	4.7	34%	47.3	6.3	12%
50,000	39.2	31.2	46.0	6.8	17%	30.6	0.6	2%
70,000	51.1	30.0	57.3	6.2	12%	28.7	1.3	4%
100,000	74.8	28.1	81.3	6.5	9%	27.6	0.5	2%
120,000	127.2	26.0	134.8	7.6	6%	25.7	0.3	1%

Table 2: Performance of one-step ITSA* on paths found by BULB in the Rubik's Cube (with 3 million nodes in memory)

ond, ARA* (Likhachev, Gordon, & Thrun 2004) repeatedly calls WA* with a different parameter setting. Therefore, ARA* is to WA* as ABULB 2 is to BULB[4] The main limitation of these variants of WA* is that, unlike ABULB, they are not memory-bounded. Therefore, both algorithms can only solve problems that WA* can solve, not including the 48-puzzle. All other anytime heuristic search algorithms known to us are variants of depth-first search, for example (Lawler & Wood 1966; Kumar 1990; Zhang 1998), which do not scale well to general graph-search problems with many transpositions and short cycles like our benchmark domains.

Conclusion

In this paper, we describe a new approach to anytime heuristic search based on local search in the space of solution paths. The two main contributions are 1) a new local optimization algorithm called ITSA* that explores the neighborhood of a given solution path in order to find shortcuts and 2) a new anytime heuristic search algorithm called ABULB that performs a neighborhood search with restarts. ABULB uses a variant of beam search called BULB to generate an initial path and ITSA* to locally optimize this path. When a minimum is reached, ABULB repeatedly restarts BULB to obtain the next path to optimize. Our empirical study shows that this new anytime algorithm quickly and significantly reduces the solution cost in two standard benchmark domains. For example, it solves the 48-Puzzle using only the Manhattan distance heuristic in a matter of minutes, with solution costs that are no worse than twice the optimal cost.

[4]One difference between ARA* and ABULB 2 is that ARA* reuses some of the search effort of the previous WA* search to speed up the current iteration. Combining ideas in incremental search with ABULB is an interesting direction for future research.

References

Aarts, E., and Lenstra, J. 1997. *Local Search in Combinatorial Optimization*. West Sussex, England: John Wiley & Sons.

Bisiani, R. 1987. Beam search. In Shapiro, S., ed., *Encyclopedia of Artificial Intelligence*. New York : Wiley & Sons. 56–58.

Boddy, M., and Dean, T. 1989. Solving time-dependent planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 979–984.

Boddy, M. 1991. Anytime problem solving using dynamic programming. In *Proceedings of the National Conference on Artificial Intelligence*, 738–743.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence*, 49–54.

Furcy, D., and Koenig, S. 2005a. Limited discrepancy beam search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 125–131.

Furcy, D., and Koenig, S. 2005b. Scaling up WA* with commitment and diversity. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1521–1522.

Furcy, D. 2004. Speeding up the convergence of online heuristic search and scaling up offline heuristic search. Ph.D. thesis, College of Computing, Georgia Institute of Technology, Atlanta (Georgia). Available as Technical Report GIT-COGSCI-2004/04.

Gaschnig, J. 1979. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Computer Science Department, Carnegie-Mellon University. Ph. D. Thesis.

Hansen, E., and Zilberstein, S. 1996. Anytime heuristic search: Preliminary report. In *Proceedings of the AAAI Fall Symposium*

B	BULB		BULB + multi-step ITSA*					
	time (seconds)	cost	time (seconds)			cost		
			value	increase over BULB		value	decrease over BULB	
				absolute	relative		absolute	relative
5	0.1	11,737	50.3	50.2	50,200%	2,562	9,175	78%
10	0.9	36,282	53.0	52.1	5,789%	1,808	34,474	95%
100	6.1	14,354	55.2	49.1	805%	1,159	13,195	92%
1,000	7.3	1,409	36.7	29.4	409%	674	735	52%
10,000	21.7	440	42.4	20.7	95%	426	14	3%

Table 3: Performance of multi-step ITSA* on paths found by BULB in the 48-Puzzle (with 6 million nodes in memory)

B	BULB		BULB + multi-step ITSA*					
	time (seconds)	cost	time (seconds)			cost		
			value	increase over BULB		value	decrease over BULB	
				absolute	relative		absolute	relative
10	96.9	108,804.8	111.3	14.4	15%	94,346.6	14,458.2	13%
100	5.1	1,893.9	23.2	18.1	356%	578.5	1,315.4	69%
1,000	7.4	275.8	22.1	14.7	199%	156.8	119.0	43%
10,000	13.8	53.6	28.1	14.3	104%	45.3	8.3	15%
50,000	39.2	31.2	49.8	10.6	27%	30.4	0.8	3%
70,000	51.1	30.0	62.4	11.3	22%	28.7	1.3	4%
100,000	74.8	28.1	86.1	11.3	15%	27.5	0.6	2%
120,000	127.2	26.0	137.4	10.2	8%	25.7	0.3	1%

Table 4: Performance of multi-step ITSA* on paths found by BULB in the Rubik's Cube (with 3 million nodes in memory)

on Flexible Computation in Intelligent Systems; Results, Issues and Opportunities, 55–59.

Hansen, E.; Zilberstein, S.; and Danilchenko, V. 1997. Anytime heuristic search: First results. Technical Report CMPSCI 97–50, Department of Computer Science, University of Massachusetts, Amherst (Massachusetts).

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Harvey, W., and Ginsberg, M. 1995. Limited discrepancy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 607–615.

Korf, R. 1997. Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of the National Conference on Artificial Intelligence*, 700–705.

Kumar, V. 1990. Branch-and-bound search. In Shapiro, S. C., ed., *Encyclopedia of Artificial Intelligence*. New York : Wiley, 2nd edition. 1000–1004.

Lawler, E., and Wood, D. 1966. Branch-and-bound methods: A survey. *Operations Research* 14(4):699–719.

Lawler, E.; Lenstra, J.; Kan, A. R.; and Shmoys, D., eds. 1985. *The Traveling Salesman Problem*. John Wiley and sons.

Likhachev, M.; Gordon, G.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of Advances in Neural Information Processing Systems 16 (NIPS)*.

Papadimitriou, C., and Steiglitz, K. 1998. *Combinatorial Optimization: Algorithms and Complexity*. Mineola, New York: Dover Publications.

Pearl, J. 1985. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pohl, I. 1970. First results on the effect of error in heuristic search. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 5. American Elsevier, New York. 219–236.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 20–23.

Ratner, D., and Pohl, I. 1986. Joint and LPA*: Combination of approximation and search. In *Proceedings of the National Conference on Artificial Intelligence*, 173–177.

Ribeiro, C., and Hansen, P., eds. 2001. *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence – A Modern Approach*. Prentice Hall, first edition.

Voss, S.; Martello, S.; Osman, I.; and Roucairol, C., eds. 1999. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic.

Winston, P. 1992. *Artificial Intelligence*. Addison-Wesley, Reading, MA, third edition.

Zhang, W. 1998. Complete anytime beam search. In *Proceedings of the National Conference on Artificial Intelligence*, 425–430.

Zhou, R., and Hansen, E. 2004. Breadth-first heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 92–100.