Some Experiments With Case-based Search*

Steven Bradtke and Wendy G. Lehnert
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

Knowedge-based problem solvers traditionally merge knowledge about a domain with more general heuristics in an effort to confront novel problem situations intelligently. While domain knowledge is usually represented in terms of a domain model, the case-based reasoning (CBR) approach to problem solving utilizes domain knowledge in the form of past problem solving experience. In this paper we show how the CBR approach to problem solving forms the basis for a class of heuristic search techniques. Given a search space and operators for moving about the space, we can use a case-base of known problem solutions to guide us through the search. In this way, the case-base operates as a type of evaluation function used to prune the space and facilitate search. We will illustrate these ideas by presenting a CBR search algorithm as applied to the 8-puzzle, along with results from a set of experiments. The experiments evaluate 8-puzzle performance while manipulating different case-bases and case-base encoding techniques as independent variables. Our results indicate that there are general principles operating here which may be of use in a variety of applications where the domain model is weak but experience is strong.

1 Introduction

Case-based reasoning (CBR) systems have been designed to address a variety of task orientations including diagnostic reasoning, adaptive planning, hypothesis generation, explanation, adversarial reasoning, analogical reasoning, and hypothetical reasoning [Rissland, 1987]. Traditionally, CBR techniques are invoked when a domain is characterized by problems that do not have right or wrong answers as much as answers that are strong or weak along various dimensions. When a novel problem is encountered, a case base of previously encountered problems and solutions is consulted to determine what experiences are relevant to the current situation. Solutions from more than one case may be merged to address the current problem, and multiple solutions are typically generated with an assessment of their respective strengths. If external feedback is provided to the system, newly solved problems can be added to the case base to strengthen it, thereby realizing a form of knowledge acquisition that is qualitatively distinct from the knowledge engineering techniques traditionally associated with rule-based systems.

In an effort to test the boundaries of CBR technology, we have applied CBR to a classic problem in heuristic search: the 8-puzzle. We have demonstrated that a heuristic search for the 8-puzzle can be conducted by accessing nothing more than a case base of previous problem solutions. For this application, the problem solutions consist of board sequences that take us from an arbitrary 8-puzzle problem state to a final goal state using legal 8-puzzle operators. No additional knowledge about subgoals [Korf, 1985], chunking [Laird *et al.*, 1987; Laird *et al.*, 1984], or any other form of derivational abstraction [Carbonell, 1986; Carbonell, 1983] is used.

We further wanted to ask questions about the construction of an effective case base, and the techniques used to index available cases in memory. Is it possible to optimize a case base? Or customize effective indices for a given case base? How can learning curves be influenced by indexing techniques or initial case bases? Although space limitations prohibit us from reporting all of our experimental results, we will describe a few of our experiments, some of which were suggested by a preliminary investigation [Lehnert, 1987]. We will also describe an index that makes it possible to generate optimal solutions for any solvable 8-puzzle state from a case base containing a single case of 31 moves.

2 Case-based Search

We have implemented a Case-Based Search algorithm (CBS) which attempts to transform a given problem state into a targeted final state by copying past problem solving performance. Each case in the case-base is a list of problem states from a start state to a goal state. Figure 1 shows one (very short) case consisting of four states. Note that if we were to remove the start state from any given case, we would be left with a new case showing the solution from a new start-state to the same goal state. Thus each case implicitly contains many others as sub-cases.

CBS has a number of operations for transforming problem states. But it should only choose *good* operations, that is, operations that transform problem states into states closer to the goal state. The case-base is intended to help

*This research was supported by an NSF Presidential Young Investigators Award NSFIST-8351863, DARPA contract N00014-87-K-0238, and the Office of Naval Research under a University Research Initiative grant, contract N00014-86-K-0764.

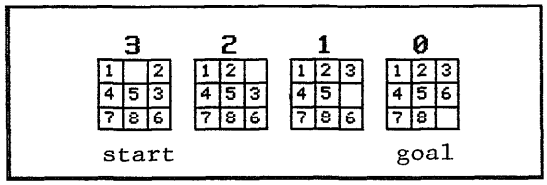

Figure 1: A Solution Case for the 8-puzzle.

the system find good operations and operation sequences. However, the case-base cannot be used to store solutions for all possible start states. Even the simple 8-puzzle has 181,440 legal board positions. CBS must be able to generalize from the solutions it finds in its case-base.

Generalization from old solutions to new solutions is done in three steps.

1. CBS uses a *coarse index function* to encode the cases in the case-base. The coarse index function maps problem states onto a set of integers (or symbols), dividing the problem states into equivalence classes. It is possible for different problem states, and thus different cases (sequences of problem states) to be mapped to equivalent coarse-coded representations. In particular, it is possible for the coarse index function to place a number of problem states in the same equivalence class as the goal state. This is not a problem as long as there is a known path from each goal-equivalent state to the goal state. These paths can be pre-computed and automatically appended to any solution found by CBS as needed. The coarse index function acts like a partial pattern matcher, relaxing similarities between structures (problem states) at the risk of allowing inappropriate matches.
2. Case-base solutions implicit in the coarse-coded case-base must be made explicit. This can be done efficiently by organizing each case in the case-base within a discrimination net. Figure 2 shows a coarse-coded case-base containing five cases, and the discrimination net that results. Every path from the dummy root node to a goal node represents a solution. Every node with no children is a goal node.
3. The final step in generalizing from old solutions to new solutions occurs during the actual search conducted for a given problem. CBS restricts exploration of the problem space by using the discrimination net described above and a masking procedure. The mask works by overlaying the discrimination net of old solutions on top of the search space generated by the given initial state. Any branches not allowed by the discrimination net are then pruned from the tree.

This masking process is best described by example. Figure 3 shows a portion of the search space reachable from a designated start state. Each node on the tree has been labelled with its coarse index value. (The actual indexing function used is not important at this time.) The tree has already been pruned so that no node is shown twice.

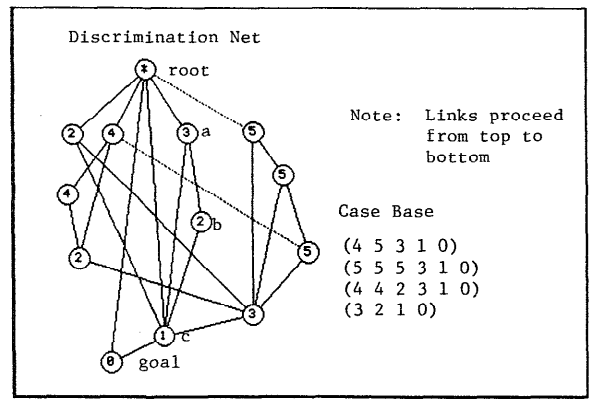

Figure 2: A Case-base and Corresponding Discrimination Net.

Nodes marked (a), (b), (c), and (goal) in figures 2 and 3 correspond to one another as the mask overlays the search space and the discrimination net. Example:

1. Take the index value of the current problem state. The index value of the start state is 3. Since the root node of the discrimination net has a child with index 3, move a marker from the root node to that child, labelled (a).
2. Look at the indices of the states reachable from the start state in the search space. These are 3, 2, and 3. But the only transitions allowed from our current position in the discrimination net are to states with indices 1 or 2. Therefore, the successor states with indices 3 are pruned from the search space. Move the discrimination net marker to node (b), and reset the current search space problem state to the successor with index 2.
3. Continue in this manner, moving down both the discrimination net and the problem search tree until we reach the goal node, or until we reach a point from which we cannot advance. If we reach a stuck state, we backtrack and continue until the entire search space is exhausted.

3 Overcoming an Inadequate Case-base

If we apply the mask to a search space and find a solution, we are done. But it is possible that the experience captured in the case-base is inadequate to solve the current problem. Then the mask-directed search will fail. In this event, standard heuristic search techniques can be used to move from an unsolvable initial problem state to a new problem state that, we hope, *is* solvable using the current case-base.

In CBS, we have implemented two simple heuristics to assist inadequate casebases. The first is a "near-miss" heuristic based on neighborhoods within the search space.

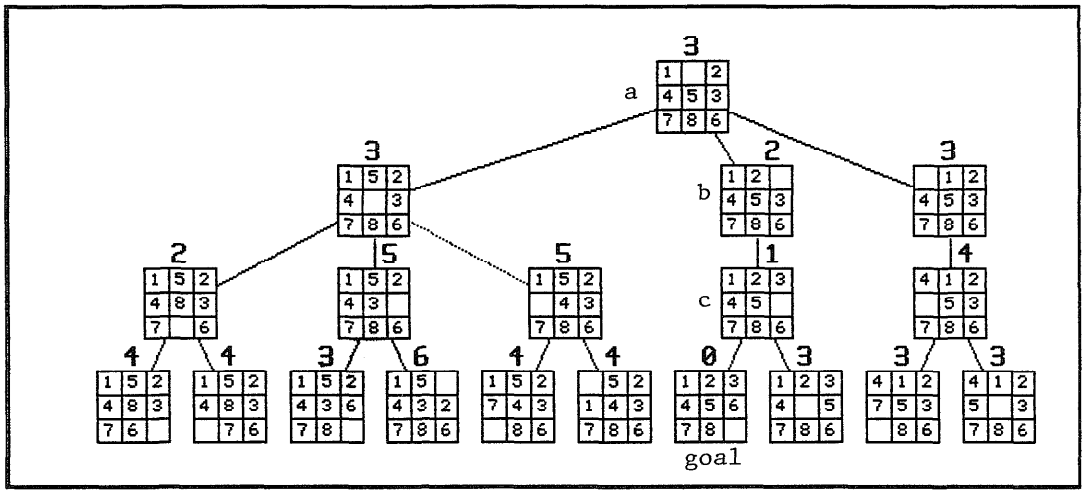

Figure 3: The Search Space Beneath a Given Problem State.

Let us define the N-family of a state to be the set of all states that can be obtained by application of N or fewer operators. If the masked search fails to produce a solution for the initial state, we then execute additional searches for each element of the initial state's N-family until either (1) a solution is found, or (2) the N-family is exhausted without success.

The second, "far-miss", heuristic is applied if the near-miss heuristic fails. Suppose we have conducted a near-miss search on the N-family of a state, and no solution has been located. Rather than extend the near-miss search into the (N+1)-family or (N+2)-family (increasing the size of the near-miss search space exponentially with each expansion), we apply a different modification to the initial state in order to generate a new search space. Given the initial state, we take a random walk of M legal moves. The resulting state will now serve as the basis for continued search. We first apply the case-based search to the new state, and if this search fails, we then apply an N-family near-miss search with the hope that our luck will be better in this region of the state space. We can keep alternating between the near-miss and far-miss heuristics until we have found a solution or we terminate at some predefined cutoff point.

4 Experiments

We ran a series of experiments to test CBS's performance on the 8-puzzle, given a variety of case-bases and coarse index functions. Each experiment tests system performance on finding solutions to 1000 randomly selected initial boards. The parameters M and N from the near-miss and far-miss heuristics were set to 10 and 4 for all experiments. The (mask, near-miss heuristic, far-miss heuristic) cycle was repeated until a maximum of 200 boards derived from the initial board via the near and far miss heuristics were looked at.

The cases-bases used in all experiments are summarized in table 1. The cases in the Random case-base were generated by randomly walking away from the goal state for an average of fifty moves, and then reversing the sequence of boards visited. The random walk was constrained to never repeat a state. The cases in the Human case-base were generated by presenting random board positions to a human player and recording the board sequences resulting from her solutions. The human player followed the usual strategy of breaking the problem into a sequence of subgoals, each to move the next tile into position while preserving the solutions to the previous subgoals [Korf, 1985]. The cases in the Perfect case-base were generated by choosing a random board and then generating and saving a minimum length solution from that position. The Random-2, Human-2, and Perfect-2 case-bases were generated in the same way, except that we tried to keep the number of *unique* boards equal instead of the *total* number of boards.

Case-base	# of cases	Total # of boards	# of unique boards
Random	21	1002	898
Human	23	1002	662
Perfect	45	1002	731
Random-2	23	1160	1009
Human-2	31	1514	1015
Perfect-2	65	1492	1016

Table 1: The case-bases.

We used seven different index functions, summarized in table 2. The city-block index function computes the city-block or "Manhattan" distance from the current board to the goal board. This index has frequently been used in previous studies of heuristic search techniques using the

Index function	# of classes	size of goal class
City-block	13	1
Binary City-block	10	2
Quad City-block	9	4
Adjacency	12	1
Relaxed Adjacency	12	4
Toroidal Adjacency	15	9
Relaxed Toroidal Adjacency	11	36

Table 2: The index functions.

8-puzzle [Nilsson, 1980]. The binary city-block index is a generalization. It computes the minimum distance from the current board to the goal board and to the 180° rotation of the goal board. The quad city-block index further generalizes the city-block index by computing the minimum distance from each of the four possible rotations of the goal board.

The four adjacency indices are based on a comparison between the neighbors each tile has in the current board and the neighbors each tile would have in the goal board. Different definitions of "neighbor" give rise to the different indices. The neighborhood of a tile under the adjacency index consists of those tiles to the right and below. The neighborhood under the relaxed adjacency index consists of those tiles to the right, left, above, and below. The neighborhood under the toroidal adjacency index is the same as for the basic adjacency index, but the board is placed on a torus, so that the first row is below the third row and the first column is to the right of the third column. The neighborhood under the relaxed toroidal adjacency index is the same as for the relaxed adjacency index, but also on a torus.

5 Results

Table 3 summarizes CBS's problem solving performance over two sets of 21 experiments, each matching one of the coarse index functions against one of the Random, Human, and Perfect (Random-2, Human-2 and Perfect-2) case-bases. The parenthesized numbers are the results of the second set of experiments. System performance was measured on two criteria: the number of problems solved (out of 1000), and the average number of boards that had to be considered before finding a solution.

The 1000 test boards were randomly and independently selected from the set of all possible 8-puzzle boards. Assuming, then, that we have chosen a representative test set, standard statistical analysis shows that we can be 95% certain that CBS's performance over the entire 8-puzzle problem space lies within 3% of the results given in the "# solved" column of table 3[1].

Analysis of the first set of experiments reveals two things.

[1]Dennis Kibler and David Ruby have been independently investigating the properties of case-based search algorithms. We would like to thank them for suggestions on possible index functions for the 8-puzzle, and for advice on statistical significance.

Index	Case-base	# solved	Avg. # searches
City Block	Random	598 (628)	87.2 (86.5)
	Human	385 (487)	88.1 (87.0)
	Perfect	533 (596)	87.9 (82.7)
Binary City Block	Random	827 (863)	67.3 (65.3)
	Human	639 (770)	80.6 (76.8)
	Perfect	775 (849)	73.3 (68.5)
Quad City Block	Random	939 (943)	55.5 (46.8)
	Human	846 (914)	66.1 (57.9)
	Perfect	906 (932)	62.2 (55.7)
Adjacency	Random	485 (486)	93.1 (88.3)
	Human	305 (375)	91.1 (84.1)
	Perfect	402 (483)	97.1 (90.5)
Relaxed Adjacency	Random	871 (886)	66.4 (64.4)
	Human	689 (841)	72.9 (70.6)
	Perfect	809 (899)	74.8 (63.9)
Toroidal Adjacency	Random	708 (699)	79.7 (79.2)
	Human	581 (668)	88.7 (79.3)
	Perfect	617 (745)	82.3 (79.0)
Relaxed Toroidal Adjacency	Random	1000 (1000)	21.9 (21.0)
	Human	998 (1000)	31.5 (24.9)
	Perfect	1000 (1000)	26.5 (18.3)

Table 3: Experimental Results.

First, as might be expected, CBS's performance depends on the number of goal equivalent states under the current coarse index function, but not all indices with the same number of goal equivalent states yield the same performance. Consider CBS's performance for a given case-base and across the coarse index functions from the city-block group. The number of problems solved rises as we move from the city-block to the binary city-block to the quad city-block index, and the average number of searches falls. Notice that the number of goal states varies from 1 to 2 to 4. The results aren't so clean cut within the adjacency index group. The overall trend matches that within the city-block group, except that the relaxed adjacency index (with 4 goal states) leads to better performance than the toroidal adjacency index (with 9 goal states). This can be explained to some extent by noticing that the relaxed adjacency and the toroidal adjacency indices are different *kinds* of generalizations upon the basic adjacency index, while the binary and quad city-block indices are the same *kinds* of generalizations.

Second, we hypothesize that CBS's performance depends on the number of unique problem states represented in the unencoded case-base. Consider, for example, CBS's performance using the city-block index. CBS does better as we move from the Human (662 unique boards) to the Perfect (731 unique boards) to the Random (898 unique boards) case-base.

We performed the second set of experiments to test this hypothesis. The results are given in parentheses in table 3. CBS's performance for a given index function is now much more equal across the three case-bases. The remaining variations in performance may be ascribed to a combination of two factors. First, it may be that the different case-bases are more or less efficient in encoding problem solving

information. The cases in the Human case-bases are constructed following an algorithm that quickly moves into a small area of the search space. It seems, then, that the human case-base would encode less of the problem solving strategy for this domain. Second, our far-miss heuristic introduces a random element which would account for some of the variation.

The striking performance of the relaxed toroidal adjacency appears to correlate with its relatively high number of goal states (36 goal states vs. an average of 3.5 goal states for the other indices). As long as we have a finite-table lookup routine that can direct us home from each of these 36 boards, we are fine. Indeed, one could argue that the overhead required to handle 36 boards is not significantly greater than the overhead associated with 4 boards, especially in view of the dramatic reduction in the number of searches required by this index. Without question, the relaxed toroidal adjacency index is superior to all other indices tested.

6 The "Perfect" Index

The "perfect" index function maps every input problem state onto a number that encodes the minimum number of moves required to transform the problem state into a goal state. Two things are apparent. First, a perfect index function needs considerable knowledge about the problem space and the current goal state. Second, access to a perfect index allows the case-based search algorithm to derive perfect hill-climbing solutions for an arbitrary problem, given an adequate case-base. Surprisingly, the minimal adequate case-base consists of a single case! That case transforms the problem state furthest from the goal into the goal state using the minimum number of operations[2]. A proof is simple. Under the perfect index, no state can have an index that differs from that of any of its neighbors by more than 1. Also, every state but the goal must have a neighboring state with a lesser index (otherwise it would have no path to the goal). Note that it is not necessary for each state have a neighbor with a *greater* index. Under the perfect index, there may exist local maxima, but *no* local minima. Thus, any perfect case of maximum length, when abstracted by the perfect index, will contain the abstraction of every other possible minimum length solution, each of which is but a step-by-step downhill march.

Table 4 summarizes the results of an experiment testing the minimum perfect case-base hypothesis, which is shown to be correct. We used the function derived in [Utgoff and Saxena, 1987] as the perfect index for the 8-puzzle. CBS displayed perfect performance while using the minimum case-base. It performed nearly as well using the Perfect case-base described in table 1. The difference in the average number of searches can be explained by the fact that it had to resort to the near-miss heuristic several times in order to find a solution. This happened because no case in the Perfect case-base covered several particularly difficult boards. It was initially surprising to see CBS perform nearly as well using the Random and Human case-bases. Again, the average number of searches was somewhat higher. In retrospect, these results are not really surprising, because they, too, follow from the argument for the existance of a minimum perfect case-base. The argument follows. Take an arbitrary solution path and encode it using the perfect index. The index profile for this case may ineffeciently wander up and down hill, but it will eventually reach zero, the index of the goal state. Since the index of every state can vary from those of its neighbors by no more than 1, any state whose index appears somewhere on the arbitrary solution path will be solvable by following a path with exactly the same index profile of the arbitrary solution, though the sequence of operators employed may be quite different.

Case-base	# solved	Avg. # searches
Random	1000	1.01
Human	1000	1.85
Perfect	1000	1.01
Minimum	1000	1.00

Table 4: Performance of the perfect index.

7 Conclusions

Given the results reported here, it appears that the most dramatic factor influencing the effectiveness of a case base is the number of unique problem states underlying the case base encoding. This explains why a case base of humanly-generated 8-puzzle solutions should be weaker than a case base of randomly generated solutions since people tend to solve the 8-puzzle by laying in the first row first, and then dealing with the remaining pieces. Once one border for the 8-puzzle has been solved, we reduce the coverage of the case base by a factor of roughly 6. This reduction is further exacerbated by the fact that the remaining boards are not distributed evenly throughout the potential search space.

In comparing competing indices, it seems that indices can trade-off different kinds of overhead against their masking hit rates. The relaxed toroidal adjacency index illustrates one such successful trade-off. Here we attain superior performance by increasing the postprocessing needed to handle 36 goal states. Since the postprocessing increase is neglible, the trade-off is quite successful.

It may also be possible to incorporate varying degrees of domain knowledge in an index. For example, the perfect index requires an extreme amount of domain knowledge (knowledge of the optimal solution lengths for all possible start states). In return, it guarantees optimal performance as long as it has access to a case base that covers the maximal solution length (31 moves for the 8-puzzle). Once domain knowledge for the perfect index has been compiled, the operation of CBS is trivial, but a substantial preprocessing overhead is required to compute this index.

Another computational trade-off can be found when we examine the computational complexity of the CBS algorithm against the overhead of a growing case base. On

[2] Assuming that there *is* a maximally difficult problem. CBS will have difficulties performing in domains where there is no bound on the possible distance of a problem from the goal. Some mechanism to allow looping, as described in [Cheng and Carbonell, 1986], is needed.

the one hand, it is computationally more expensive to exhaust a larger case base (which happens whenever we fail to find a solution). On the other hand, the chances of finding a solution increase as the case base grows larger (and a successful search terminates before the case base is exhausted). Additional experiments have been conducted to examine this trade-off, and those results show that the computational effort associated with successful CBS executions remains constant as new cases are added to the case base [Ruby and Kibler, 1988].

In closing, we must note that CBS is not a good prototype for CBR system development in general. Because the 8-puzzle has so little domain knowledge associated with it, CBS is strictly limited to a heuristic search algorithm. This makes it impossible to merge multiple solutions from the case base, or generate multiple solutions to the current problem state that can be compared in interesting ways. Unlike most other CBR applications, answers to an 8-puzzle problem *are* either right or wrong. We must also point out that CBS cannot make any claims about psychological plausibility, whereas most CBR systems are inspired by techniques presumed to be useful in human problem solving.

While the results reported here may not provide answers to the most compelling problems of CBR system development in domain-rich applications, we believe we have made a contribution to the CBR research effort by showing how general the CBR approach to problem solving really is. We have successfully applied CBR to a classic problem in heuristic search, and have therefore extended the range the potential CBR applications beyond their previous scope.

References

[Carbonell, 1983] J. G. Carbonell. Learning by analogy, formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, chapter 5, Tioga Publishing Company, Palo Alto, CA, 1983.

[Carbonell, 1986] J. G. Carbonell. Derivational analogy: a theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, vol. 2*, chapter 14, Morgan Kaufmann, San Mateo, CA, 1986.

[Cheng and Carbonell, 1986] P. W. Cheng and J. G. Carbonell. The FERMI system: inducing iterative macro-operators from experience. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 1986.

[Korf, 1985] R. E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[Laird *et al.*, 1987] J. Laird, A. Newell, and P. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[Laird *et al.*, 1984] J. Laird, P. Rosenbloom, and A. Newell. Towards chunking as a general learning mechanism. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 1984.

[Lehnert, 1987] W. G. Lehnert. *Case-Based Reasoning as a Paradigm for Heuristic Search.* COINS 87-107, Department of Computer and Information Science, University of Massachusetts at Amherst, Amherst, MA, 1987.

[Nilsson, 1980] N. J. Nilsson. *Principles of Artificial Intelligence*, page 85. Tioga Publishing Company, Palo Alto, CA, 1980.

[Rissland, 1987] E. L. Rissland. *Research Initiative in Case-Based Reasoning.* CPTM 17, Department of Computer and Information Science, University of Massachusetts at Amherst, 1987.

[Ruby and Kibler, 1988] D. Ruby and D. Kibler. Exploration of case-based problem solving. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, Morgan Kaufmann, San Mateo, CA, 1988.

[Utgoff and Saxena, 1987] P. Utgoff and S. Saxena. *A Perfect Lookup Table Evaluation Function for the Eight-Puzzle.* COINS 87-71, Department of Computer and Information Science, University of Massachusetts at Amherst, 1987.