University of Alberta

Efficiently Searching the 15-Puzzle

by

Joseph C. Culberson and Jonathan Schaeffer

Technical Report TR 94–08
May 1994

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

Efficiently Searching the 15-Puzzle

Joseph C. Culberson Jonathan Schaeffer

Abstract

The A* algorithm for single-agent search has attracted considerable attention in recent years due to Korf's iterative deepening improvement (IDA*). The algorithm's efficiency depends on the quality of the lower bound estimates of the solution cost. For sliding tile puzzles, reduction databases are introduced as a means of improving the lower bound. The database contains all solutions to the subproblem of correctly placing N tiles. For the 15-Puzzle, IDA* with reduction databases (N=8) are shown to reduce the total number of nodes searched on a standard problem set of 100 positions by over 1000-fold. With the addition of transposition tables and endgame databases, an improvement of over 1700-fold is seen.

1 Introduction

In recent years, the A* algorithm for single-agent search has attracted considerable attention. Korf pioneered the use of iterative deepening with depth-first A* (IDA*), eliminating many of the difficulties of the basic A* algorithm [4]. Even with this enhancement, the search trees built may be quite large. To make the search more efficient, one can look for algorithm improvements (such as removing duplicate nodes from the search [11, 10]), tighter lower-bound estimates on the solution cost (for example, the linear conflicts heuristic for sliding-tile puzzles [2]) or consider a hybrid version of A* that produces "good" non-optimal solutions (for example, real-time search [5], linear-space best-first search [6]). However, with the constraint of requiring an optimal solution, few improvements to the basic IDA* algorithm have been proposed. Given that machines with gigabytes of RAM will soon be commonplace, are there ways of using memory to dramatically decrease the size of the search tree?

This paper discusses using a large memory to reduce the cost of searching sliding tile puzzles. The 15-Puzzle is illustrated in Figure 1. The puzzle consists of a set of 15 *labeled cells* arranged in a 4×4 square grid, with one grid location remaining *empty*. The empty location is called the *empty cell.* We refer to the cells by their labels $0 \leq i \leq 15$, where 0 indicates the empty cell. A move consists of sliding a tile into the empty square. The object is to reach a goal state in the minimum number of moves.

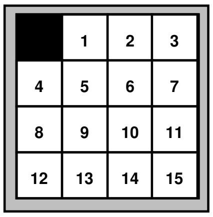

Figure 1: The 15-Puzzle Goal.

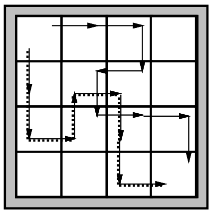

Figure 2: Mirror Path Reflection.

The problem can be generalized into an $n \times m$ puzzle. The 2×2 puzzle is trivial with a search space isomorphic to a 12-cycle. The 3×3 puzzle has been completely solved [8]. The search space for the 15-Puzzle has $16!/2 = 10,461,394,944,000 \approx 10^{13}$ positions. It is well known that one half of the permutations cannot be reached from the goal, and that these can be detected with a simple parity test [3]. Work has begun on trying to find efficient algorithms for solving the 5×5 puzzle ([6], for example). It is known that determining whether a path of a given length from an arbitrary position to the goal exists is NP-hard for the general $n \times n$ puzzle [7].

In two-person game-playing programs, there are two popular techniques for trading memory for time that can also be applied to single-agent search. First, transposition tables are used to record the result returned by the search of each subtree. In the event that another search path transposes into a saved line, the value of the subtree can be retrieved from the table, allowing reuse of the previously computed result. Marsland and Reinefeld have shown that this technique can reduce the size of a search tree in half [10], but Taylor and Korf have shown that finite state machines can do better [11].

Second, endgame databases are used to record a subset of states in the search space that are close to the solution state(s). In two-person game-playing programs, the usual method is to compute the value (win/loss/draw) for all positions with N or fewer pieces on the board. When the search reaches a position with N or fewer pieces, the value from the database is retrieved and further search along that path is unnecessary. In single-agent search, all positions in the search space within a distance of N of the solution can be saved. To the best of our knowledge, results have not been reported showing the effectiveness of this technique on single-agent search.

Given a gigabyte of RAM, are these techniques the most effective way to use the memory? This paper introduces *reduction databases* as a new approach for solving sliding tile puzzles (the 15-Puzzle) using IDA*. A human-like approach to solving the problem non-optimally is to move some of the tiles into their correct position, thereby reducing the complexity of the remaining problem to be solved. A reduction database contains the minimal number of moves required to correctly place N designated tiles. Given an arbitrary position, the positions of the N designated tiles are used to index into the reduction database to find the minimal number of moves required to correctly place those N tiles. Further, because of symmetries in the puzzle (mirror, horizontal, vertical and diagonal) a position can be mapped to alternate images, each of which can be looked up in the reduction database. The maximum over all the database lookups results in a lower bound on the solution cost. Essentially, a reduction database can be viewed as a collection of solutions to subproblems.

Gasser has also developed a technique for using memory to reduce the cost of solving sliding tile puzzles [1]. He uses a database of patterns to achieve a 50-fold reduction in the tree sizes. Patterns consist of combinations of up to 4 tiles. For each pattern the database contains the minimum number of moves required to maneuver the tiles into their correct positions, ignoring the constraints of the blank tile. Essentially, these databases improve the Manhattan distance metric by taking into account that movement of some of the tiles will conflict with each other.

In this paper, the results of using reduction databases, transposition tables and endgame databases are reported for solving the 15-Puzzle. On a standard set of 100 test positions, reduction databases reduce the total number of nodes over all the problems by 1038-fold compared to using just the Manhattan distance. The addition of transposition tables and endgame databases improve this to a 1707-fold reduction in the search trees built.

2 Database Fundamentals

A *position* is a permutation of the cells with respect to the *goal* or identity permutation τ as indicated in Figure 1(a). Specifically, for a position p, $p[i]$ is the cell in location i. The permutation for a position is obtained by a left to right, top to bottom traversal of the puzzle. Given permutations a and b, $c = a[b]$ is the permutation obtained by $\forall i, c[i] = a[b[i]]$ and $c[a] = b$ is the permutation obtained by $\forall i, c[a[i]] = b[i]$. We can compute the inverse a^{-1} of a by $a^{-1}[a] = \tau$. $c[b[a]] = (c[b])[a]$ for any a, b, c which we abbreviate as abc. Also, $c[b] = a$ and $c = a[b^{-1}]$ are equivalent definitions of c.

In any position, a *move* consists of moving one of the orthogonally adjacent cells into the empty location. However, we prefer to think of a move as swapping the empty cell with one of its neighbors, and will specify a move by the direction the empty cell moves, with l, r, u, d corresponding to *left, right, up* and *down* respectively. Two positions are adjacent in the puzzle space if one can be obtained from the other in a single move. The *distance* between two positions is the minimum number of moves required to produce one from the other. The *cost* of a position is the distance from the goal to the position. A *path* $P(\tau, p)$ from one position (p) to another is a sequence of moves (permutation τ) transforming the first into the second. Note that for each path there is a corresponding sequence of positions.

2.1 Mirror Positions

For every path $P(\tau, p)$ there is an equal length path $P'(\tau, p')$ obtained from P by reflecting the moves across the main diagonal. P' is obtained from P (or vice versa) by the replacements $l \leftarrow u, u \leftarrow l, r \leftarrow u, u \leftarrow r$ for each move in P. We call P' the *mirror path* of path P and p' the *mirror* of position p. Figure 2 illustrates the path reflection.

Let m be the permutation that reflects the puzzle across the main diagonal. Thus, $m = (0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15)$. By induction, if the kth move of the path P moves cell i to location j, then move k of the path P' moves cell $m[i]$ to location $m[j]$. Thus, the image (under m) of the ith cell of a position p will be in the $m[i]$th position of the mirror p', or in shorthand $m[p[i]]$ will be in $p'[m[i]]$. So p' is defined by $p'[m] = m[p]$. The *mirror* p' is then a reflection and permutation of p across the main diagonal.

Since for every path $P(\tau, p)$ there is an equal length path $P'(\tau, p')$ and vice versa, it follows that

Lemma 2.1 *Any lower bound on the cost of a position p or its mirror p' applies to both p and p'.*

These facts can be used to great advantage while searching for the cost of a position. In databases, we need only store information about a position and not its mirror, resulting in a nearly 50% reduction. (We get slightly less than 50% because some positions are self-mirrors, for example the goal position).

2.2 Reduction Databases

An *image* is the partial specification of a permutation (or position). That is, the cells occupying certain locations are unspecified. These unspecified cells are called *blanks*. Note the distinction between blank cells and the empty cell. In all of the images we use in this paper the empty cell will be specified.

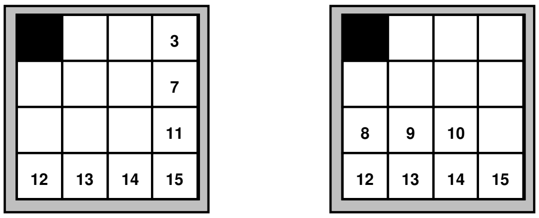

Figure 3: The Fringe and Corner Target Images.

A *target image* is a partial specification of the goal position. A *reduction database* or *image database* (RDB) is the set of all images which can be obtained by permutations of a target image. Two target images for which we have built databases are shown in Figure 3. We refer to these databases as the *fringe* and the *corner* respectively.

For each image in a database, we compute the distance (minimum number of moves) to the target image. We refer to this distance as the *cost of the image*. Note that the distance includes the cost of moving the blank cells when required to place the specified cells, but does not require the blanks to be in any particular order. Since the specified cells are in their final locations in the target image we have

Lemma 2.2 *For any position, for any image database, the cost of the image induced by the position with respect to the database is a lower bound on the cost of the position.*

In addition to lower bounds, we can also obtain upper bounds from an RDB. We will consider the fringe database as an example, but similar arguments apply to any RDB.

In addition to the cost of obtaining the fringe target image, we store the permutation associated with one optimal move sequence that generates it. Then, for any position, applying that permutation results in a new position, with only the 3×3 subpuzzle remaining to be solved. Since the solution of the 8-puzzle is easily computed, adding the two costs together yields an upper bound on the total cost.

By modifying this RDB slightly, we can also obtain additional lower bounds and a tighter upper bound. Instead of a single fringe target image, we consider the set of nine targets in which the empty cell is located in one of the upper 3×3 locations of the puzzle, and the specified cells are the same as in the fringe target image.

We compute the cost of solving each position of the 8-puzzle by searching it in the 15-puzzle. In this case, 172 positions have a reduced cost of 2 over the standard 8-puzzle. This gives us a tighter upper bound in some cases.

To obtain a tighter lower bound let f be the cost of reaching a target image in this database, and r be the cost of solving the remaining 8-puzzle. Consider any other path from the current position to the goal, with cost to a fringe image f' and 8-puzzle cost of r'. Since r is a minimum cost solution of the first 8-puzzle subgoal within the 15-puzzle, then $r \leq f + f' + r'$, and thus $f' + r' \geq r - f$. When $r > 2f$, this gives a better lower bound than f. If it is feasible to wait until the search reaches at least once to a position in which the target image is realized, then it is not necessary to store the permutation to use this improved lower bound. Instead, the result can be backed up in a depth first search.

However, both the upper bounds and the improved lower bounds have almost no effect when the other databases and simplifications are included in the IDA* search. In addition, the storage required for the permutations is significant. For these reasons, these techniques are not of much practical benefit.

2.3 Using Symmetry

Using the properties of mirror positions discussed earlier, it is possible to obtain two lower bounds from a database, one for p and one for p'. In this case, the mirror of a blank cell is assumed to be blank. Notice, however, that due to the symmetry of the fringe target image, the cost of an image and its mirror are always equal in the fringe RDB.

Using other symmetries of the 15-Puzzle, even better lower bounds can be obtained. Let v be the permutation induced by reflection about the vertical axis, h be the permutation induced by reflection across the horizontal axis and d be the permutation induced by reflection across the transverse diagonal. Consider the image in Figure 4. Given the fringe RDB, the distance to this image can be found by defining $\hat{v} = v$, with the exception that $\hat{v}[0] = 0$ and $\hat{v}[3] = 1$ and vice versa. Now define the vertical reflection of a position p by $p^v[v] = \hat{v}[p]$. Since $v = v^{-1}$ and $\hat{v} = \hat{v}^{-1}$, computing p^v for any position p and looking up p^v in the fringe RDB produces the minimum distance from p to the image in Figure 4. But any position matching the image in Figure 4 is three moves away from a target image which is obtained by moving the empty cell three steps to the left. Note in particular that this move sequence leaves cell 1 in its correct location.

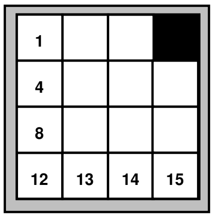

Figure 4: Image Reflection Across the Vertical Axis.

A similar argument holds for any RDB. Thus, a lower bound on reaching the goal can be obtained by computing p^v for any position p, looking up the cost in the RDB and subtracting three.

A similar case holds for the reflection h.

For d, the initial reflection carries the empty cell to the lower right corner. This requires 6 moves of the empty cell to create a target image. Thus, it is necessary to subtract 6 from the RDB cost. However, there are several distinct paths (with respect to the effects on specified cells) of length six. Thus, there are several different lookups that may be performed for the reflection d.

Finally, for each of the positions obtained by reflecting across h, v, d, the mirror of the position can also be computed and looked up in the appropriate

RDB. None of these mirrors are effective in the fringe because, as noted previously, the fringe target image is symmetric with respect to the mirror.

From the preceding discussion, it might be concluded that the fringe is at a disadvantage because symmetry means the mirror never yields improved lower bounds. On one hand, we intuitively expect that the fringe database will yield somewhat tighter lower bounds on average for a single lookup. The reason is that once the fringe target image is achieved, it is quite unlikely that the specified cells will be disturbed in completing the rest of the solution. In fact, as previously mentioned, at most two moves can be saved from any position matching the fringe target image over the corresponding solution restricted to 8-puzzle moves. On the other hand, the corner target image may be significantly disturbed in order for the solution to be completed. This argument places some intuitive limits on which target images are likely to be useful. For example, a target image in which only alternate cells are specified could seemingly be very far from the goal. Such a database would likely yield weak lower bounds for most positions.

These are never-the-less intuitive arguments, and detailed analysis, either experimental or analytical, remains to be done.

3 Example

Table 1 shows the results of looking up a position in the reduction database (position number 79 in the Korf set [4], the simplest problem in the set). The table shows all the images for the original position (O): its mirror (M), horizontal (H), horizontal mirror (HM), vertical (V), vertical mirror (VM), diagonal (D) and diagonal mirror (DM). Note that there are 10 diagonal images for the corner and only 6 for the fringe (since there are more unique minimal paths for the empty cell to traverse from square 15 to square 0).

For this position, the Manhattan distance is 28. Looking up the original position in the fringe database yields a lower bound of 38 (the maximum over all the images examined). Over all 100 Korf test positions, reduction databases using the fringe improved the Manhattan distance 92 times for an average gain of 6.9 (the other 8 positions were the same as Manhattan). Using the corner database, the bound was improved 94 times with an average gain of 6.1, stayed the same 5 times, and was not as good once (by 2).

Name	Position	DB Value	Subtract	Bound
	Corner			
O	0 1 9 7 11 13 5 3 14 12 4 2 8 6 10 15	28	0	28
M	0 14 11 2 4 7 3 9 6 5 1 10 13 12 8 15	32	0	32
HO	7 10 1 0 3 5 14 8 2 4 15 13 12 9 6 11	35	3	32
HM	2 8 13 0 10 3 7 4 9 1 5 6 12 11 15 14	35	3	32
VO	8 10 6 3 2 4 12 14 11 1 9 15 0 13 5 7	37	3	34
VM	1 4 8 3 10 9 13 6 12 7 15 5 0 2 11 14	29	3	26
DO	4 5 9 7 13 15 3 1 12 10 2 11 8 6 14 0	32	6	26
DO	1 6 9 7 13 4 3 5 12 14 2 11 8 10 15 0	22	6	16
DO	1 9 13 7 14 4 3 2 12 10 6 11 8 5 15 0	32	6	26
DO	1 5 9 11 14 4 7 2 13 10 3 8 12 6 15 0	34	6	28
DO	1 9 10 7 13 4 3 5 12 14 2 11 8 6 15 0	30	6	24
DO	1 6 10 11 13 4 3 5 12 14 2 8 9 7 15 0	34	6	28
DO	1 5 13 11 14 4 3 2 12 10 6 8 9 7 15 0	36	6	30
DO	4 9 10 7 13 15 3 1 12 11 2 5 8 6 14 0	36	6	30
DO	4 6 9 7 13 15 3 1 12 11 2 5 8 10 14 0	32	6	26
DO	4 6 10 5 13 15 3 1 12 11 2 8 9 7 14 0	38	6	32
DM	4 7 3 2 5 14 10 9 6 12 8 15 13 11 1 0	30	6	24
DM	1 7 3 2 6 15 14 9 10 12 8 4 13 11 5 0	30	6	24
DM	1 7 3 6 9 15 10 13 5 12 8 4 14 11 2 0	32	6	26
DM	1 11 7 3 5 15 10 9 6 13 12 4 14 8 2 0	32	6	26
DM	1 7 3 2 9 15 14 10 6 12 8 4 13 11 5 0	30	6	24
DM	1 11 3 2 6 15 14 10 7 12 9 4 13 8 5 0	28	6	22
DM	1 11 3 6 5 15 10 13 7 12 9 4 14 8 2 0	32	6	26
DM	4 7 3 2 9 14 11 10 6 12 8 15 13 5 1 0	30	6	24
DM	4 7 3 2 6 14 11 9 10 12 8 15 13 5 1 0	30	6	24
DM	4 5 3 2 6 14 11 10 7 12 9 15 13 8 1 0	28	6	22
	Fringe			
O	0 1 9 7 11 13 5 3 14 12 4 2 8 6 10 15	38	0	38
HO	4 9 3 0 1 5 14 8 2 7 15 13 12 10 6 11	29	3	26
VO	2 8 13 0 9 1 4 7 10 3 5 6 12 11 15 14	31	3	28
DO	1 10 6 7 14 11 3 2 13 5 8 4 12 9 15 0	34	6	28
DO	1 10 13 7 14 11 3 2 12 5 6 4 8 9 15 0	38	6	32
DO	1 10 6 7 13 11 3 5 12 14 2 4 8 9 15 0	30	6	24
DO	4 10 6 11 13 15 7 1 12 5 3 8 2 9 14 0	38	6	32
DO	4 10 6 11 13 15 3 1 12 5 2 8 9 7 14 0	36	6	30
DO	4 10 6 7 13 15 3 1 12 11 2 5 8 9 14 0	34	6	28

Table 1: Position #79 and its Images.

4 Experiments

Iterative-deepening A* for the 15-Puzzle was implemented using the Manhattan distance heuristic estimate (MD), transposition tables (TT, with 2^{18} positions [10]), endgame databases (DB, all solutions of length <= 25 moves), fringe reduction database (FR) and corner reduction database (CO). The fringe and corner databases were built using retrograde analysis and each contains $16 \times 15 \times 14 \times 13 \times 12 \times 11 \times 10 \times 9$ = 518,918,400 positions, one byte each. The programs were written in C and run on a BBN TC2000 at Lawrence Livermore Laboratories. The TC2000 has 128 processors and 1 GB of RAM. The program would load the databases into shared memory and search each test position in parallel. An experiment consisted of running the 100 test positions given by Korf [4].

Table 2 summarizes the results. Transposition tables reduce the total tree size over all 100 test problems by a factor of 2.7-fold[1]. This is somewhat better than the results reported by Marsland and Reinefeld [10] (a 2-fold reduction) for two reasons: 1) the hash function used is slightly better [9] and 2) both a position and its mirror image are looked up in the table (since each position has a mirror position with an equivalent solution length, the search results for one are applicable to the other[2]). However, even with these improvements, the results are still inferior to Taylor and Korf's 3.6-fold reduction in tree size through the use of a finite state machine to detect duplicate states [11].

Endgame databases on their own reduce the search space a factor of 1.9-fold. The database of all positions with solution length of 25 consists of $36,142,146$ positions (most of the mirror positions are removed), compressed into 106 MB. Transposition tables use considerably less memory (2^{18} entries with 12 bytes/entry = 3.1 MB in these experiments) and achieve better performance results. However, increasing the size of the transposition table provides decreasing benefits, whereas increasing the size of the endgame

[1]This measurement is used for compatibility with other works. However, it is not necessarily a fair comparison, since the larger trees have a greater influence on the final result than the smaller trees.

[2]This suggests a better implementation would be to always "normalize" the board, by not allowing mirror position in the search. For example, one simple normalization that would eliminate most of the mirror positions would be to keep the empty square on or below the diagonal. If a move causes the empty square to move above the diagonal, reflect it to below the diagonal.

database provides increasing benefits. In this experiment, the combination of the two resulted in a 4.1-fold decrease in the search trees.

Table 2 shows that the fringe database reduces the search trees by 345-fold, and the corner database is even better with a 437-fold reduction. Since the fringe is symmetric, it has fewer images to look up (9) than for the corner (26). Thus, the corner's lower bound is maximized over more values, generally giving a tighter bound. When transposition tables and endgame databases are used with the corner database, a further 1.7-fold reduction is seen, yielding a total reduction of 773-fold.

Experiment	Total Nodes	Tree Size %	Improvement
MD	36,302,808,031	100.00	1.0
MD+DB	19,419,742,608	53.49	1.9
MD+TT	13,662,973,000	37.64	2.7
MD+TT+DB	8,869,627,254	24.43	4.1
MD+FR	105,067,478	0.29	345.5
MD+CO	83,125,633	0.23	436.7
MD+TT+DB+FR	52,839,191	0.15	687.0
MD+TT+DB+CO	46,987,450	0.13	772.6

Table 2: Results Summary.

Examination of the data produced by the reduction databases showed that for many positions they produced complimentary results: positions where the corner did poorly often had the fringe doing well, and visa-versa. Obviously the program could be modified to take the maximum over both databases, but since each database is 520 MB this does not seem practical. Instead, the program was modified to use half of each database. Each reduction database was broken into 16 parts - one for each square that could be empty. If the 4×4 board is viewed as a checkerboard, then for half of the squares the corresponding fringe database (F) would be read into memory and for the other half, the corresponding corner database (C) would be read (as in Figure 5a). Given a position, the images for which the appropriate reduction database is in memory are looked up. For example, Figure 5b shows the position of an empty square (O) and where the it would appear in all the images of that position. The original position (O), its mirror (M), corner diagonal (DO) and corner diagonal mirror (DM) images would be looked up using the corner database, while the fringe vertical (VO), the fringe vertical mirror (VM), the fringe horizontal (HO) and the fringe horizontal mirror (HM) images would be looked up in the fringe database. When the empty square moves once, the original position, mirror, fringe DO and fringe DM images would be looked up using the fringe database, while the corner VO, the corner VM, the HO and the corner HM images would be looked up in the corner database. Essentially, this scheme allows us to use half of each database to achieve the benefits of having both.

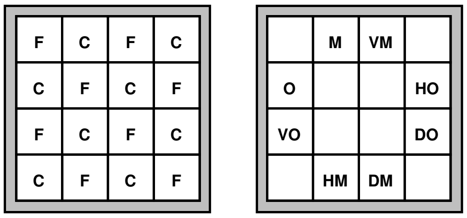

Figure 5: a) 15-Puzzle as a Checkerboard. b) Looking up a Position.

Using half of the corner and fringe databases (BT) without any other enhancements results in the tree size being reduced by 1038-fold. Tables 3 and 4 show the results of using BT combined with transposition tables and endgame databases. For comparison purposes, the 100 Korf positions and the size of their Manhattan distance search trees are shown. Using both databases gives a 2.5-fold improvement over just using the corner database, resulting in a search size that is 0.059% of the Manhattan trees, a 1707-fold reduction. The average problem has its tree size reduced by a factor of 1775.

The linear conflicts heuristic (LC) is a recently proposed improvement to the Manhattan distance heuristic [2]. It improves the Manhattan distance measure by recognizing when two tiles in the same row/column will conflict with each other moving into their correct position (a linear conflict). Tables 3 and 4 include the results of using linear conflicts with and without the databases. The databases improve this heuristic 214-fold. Note that when the memory tables are not used, linear conflicts reduces the Manhattan trees by 9.7-fold. With the database, a small improvement of 1.071 is seen, implying most of the benefits of linear conflicts are captured in the reduction databases.

#	MD	MD+TT+DB+BT	Improvement	LC	LC+TT+DB+BT	Improvement
1	276361933	28977	9537	12205622	28478	429
2	15300442	80054	191	4556066	76408	60
3	565994203	682529	829	156590305	654265	239
4	62643179	40798	1535	9051743	39675	228
5	11020325	124746	88	2677665	111686	24
6	32201660	50682	635	4151681	49454	84
7	387138094	370549	1045	97264709	365079	266
8	39118937	47756	819	3769803	45889	82
9	1650696	4411	374	88587	3106	29
10	198758703	584055	340	48531590	564714	86
11	150346072	259862	579	25537947	249039	103
12	546344	3170	172	179627	3026	59
13	11861705	6740	1760	1051212	6370	165
14	1369596778	71892	19051	53050798	70868	749
15	543598067	387056	1404	130071655	374282	348
16	17984051	14532	1238	2421877	14500	167
17	607399560	411001	1478	100843885	356736	283
18	23711067	9440	2512	5224644	9045	578
19	1280495	4773	268	385368	4502	86
20	17954870	78997	227	3642637	72445	50
21	257064810	239410	1074	43980447	236112	186
22	750745755	434487	1728	79549135	409107	194
23	15971319	25659	622	770087	19785	39
24	42693209	73051	584	15062607	72213	209
25	100734844	49508	2035	13453742	48938	275
26	226668645	261722	866	50000802	255581	196
27	306123421	56465	5421	31152541	55821	558
28	5934442	2837	2092	1584196	2733	580
29	117076111	59776	1959	10085237	54377	185
30	2196593	2637	833	680253	2591	263
31	2351811	1128	2085	538885	1127	478
32	661041936	2176909	304	183341086	1969359	93
33	480637867	367663	1307	28644836	333084	86
34	20671552	59518	347	1174413	51037	23
35	47506056	4673	10166	9214046	4660	1977
36	59802602	56116	1066	4657635	54648	85
37	280078791	187652	1493	21274606	181121	117
38	24492852	26340	930	4946980	25392	195
39	19355806	29233	662	3911622	29065	135
40	63276188	92937	681	13107556	91043	144
41	51501544	100283	514	12388515	94749	131
42	877823	2219	396	217287	2159	101
43	41124767	187585	219	7034878	147917	48
44	95733125	51932	1843	3819540	46438	82
45	6158733	14116	436	764472	13724	56
46	22119320	21065	1050	1510386	19465	78
47	1411294	769	1835	221530	766	289
48	1905023	15243	125	255046	12608	20
49	1809933698	512255	3533	203873876	496204	411
50	63036422	164203	384	6225179	153052	41

Table 3: Results for the 100 Korf Positions (I).

#	MD	MD+TT+DB+BT	Improvement	LC	LC+TT+DB+BT	Improvement
51	26622863	69695	382	4683053	66326	71
52	377141881	203099	1857	33691152	197048	171
53	465225698	358807	1297	125641729	346544	363
54	220374385	111930	1969	26080658	109220	239
55	927212	1447	641	163076	1442	113
56	1199487996	314872	3809	166183824	311350	534
57	8841527	10241	863	3977808	10105	394
58	12955404	21647	598	3563940	21303	167
59	1207520464	148213	8147	90973286	146131	623
60	3337690331	3429574	973	256537527	3047593	84
61	7096850	13364	531	672958	12849	52
62	23540413	32331	728	8463997	30229	280
63	995472712	494062	2015	20999335	444812	47
64	260054152	155874	1668	43522755	154996	281
65	18997681	19181	990	2444272	18526	132
66	1957191378	826670	2368	394246897	811211	486
67	252783878	118678	2130	47499461	117263	405
68	64367799	23123	2784	6959506	23060	302
69	109562359	25425	4309	5186586	24412	212
70	151042571	96851	1560	40161672	96463	416
71	8885972	3666	2424	539386	3594	150
72	1031641140	85060	12128	55514359	83436	665
73	3222276	10870	296	1130806	10431	108
74	1897728	30106	63	310311	25658	12
75	42772589	54052	791	5796659	52611	110
76	126638417	205942	615	25481595	195302	130
77	18918269	66473	285	5479396	63727	86
78	10907150	33812	323	2722094	33118	82
79	540860	2066	262	107087	1859	58
80	132945856	336552	395	39801474	314887	126
81	9982569	10445	956	1088122	10204	107
82	5506801123	825333	6672	203606264	788489	258
83	65533432	14831	4419	2155879	14475	149
84	106074303	66087	1605	17323671	65271	265
85	2725456	3460	788	933952	3459	270
86	2304426	12003	192	237465	9915	24
87	64926494	26929	2411	7928513	26857	295
88	6320047980	2611177	2420	422768850	2398192	176
89	166571021	266442	625	29171606	252407	116
90	7171137	9541	752	649590	9135	71
91	602886858	472683	1275	91220186	461788	198
92	1101072541	472553	2330	68307451	458495	149
93	1599909	2224	719	350207	2184	160
94	1337340	22823	59	390367	20005	20
95	7115967	23040	309	1517919	21854	69
96	12808564	26877	477	1157733	25096	46
97	1002927	2592	387	166565	2452	68
98	183526883	80307	2285	41564668	79903	520
99	83477694	116601	716	18038549	112525	160
100	67880056	352705	192	17778221	328188	54
TTL	36302808031	21261747		3759631279	19850843	
AVG	100.0	0.059	1774.648	100.0	0.528	214.626

Table 4: Results for the 100 Korf Positions (II).

In Table 5, the 100 positions have been sorted into groups of 20 based on the number of nodes required to solve the MD tree. The results show that the 20 easiest problems benefit by a factor of 628 (142 for LC) using databases and tables, while the 20 hardest problems benefit by a factor of 3,706 (313 for LC). This suggests that the improvements are limited only by the small problems, whose solution trees become close to the minimal tree.

Problems	MD	LC
01-20	628	142
21-40	745	116
41-60	1679	251
61-80	2116	251
81-100	3706	313

Table 5: Results Grouped by Size of Search Tree.

Obviously, the enhancements presented in this paper increase the cost of evaluating a node in the search tree. The question arises whether the additional work done by the node evaluation is outweighed by the reduction in search tree size. Our implementation of the program is simple and regenerates images at each node (as many as 26), rather than maintaining all the information incrementally. Further, the program uses the maximum number of images for computing a bound. The diagonal images contribute the least to the bound (since they have 6 subtracted from their bounds) and eliminating them will more than double the speed of the program, while only increasing the tree by a small percent. Even without these enhancements the program runs roughly 6 times faster in real time compared to MD, while it runs roughly 1.5 times faster than LC. However, using BT by itself, with only the original and mirror position queried in the reduction database, results in a larger search tree, but the program runs 12 times faster than MD in real time. There is a trade-off here: increased cost per node versus better lower bound. We have not done experiments to find which combination minimizes execution time.

5 Conclusions

Reduction databases are an effective means for significantly reducing the size of search trees in the 15-Puzzle. By using lower bounds on the cost of solving sub-problems, a heuristic that is significantly more effective than the Manhattan distance is obtained. The large reduction in search tree size does not give an equivalent reduction in the execution time required to solve the problem. This is largely due to the simplicity of the Manhattan distance heuristic, since it is inexpensive to compute. For other problems where a more computationally expensive lower bound is required, the relative efficiency of the reduction databases at run-time will improve.

Since we had access to a gigabyte of RAM, we used the largest reduction databases that we could (8 tiles). A reduction database that placed only 7 tiles would required 57 MB. The smaller databases would provide weaker lower bounds for the 15-Puzzle, but they would still be significantly better than MD.

This work can be extended to the 24-Puzzle. For the 15-Puzzle, the reduction database places 8 of the 16 tiles. Using comparable storage for the 24-Puzzle, the database will only be able to contain 6 of the 25 tiles. Because of the complexity of this problem, it is too expensive to search for optimal solutions. Instead, the effort has been in finding approximate solutions quickly [5]. The database information can be used to increase the accuracy of any heuristic estimate. The larger the puzzle, the more effective become the images. For example, the cost of placing 6 tiles on one side of the board may have little effect on the cost of placing the mirror of the 6 tiles on the other side of the board. Thus the maximum value from the reduction database of the images will decrease slowly. This suggests that a two-level heuristic estimate may be an effective real-time search strategy: given two states with the same reduction database maximum value, search the one with the smallest sum (or average) of the individual database image values. This will concentrate effort on lines which are making a greater contribution to the overall solution of the problem.

With the 8-Puzzle trivially solvable and any 15-Puzzle image being solvable with a relatively small search, there may be ways to combine these results to help obtain good bounds on the solution lengths for the 24-Puzzle. For example, the 24-Puzzle can be viewed as being composed of several overlapping 15- or 8-Puzzles. A function would have to be developed that allowed the results of these overlapping problems to be combined.

Although reduction databases have only been applied to the 15-puzzle, the idea in principle is applicable to other single-agent search domains. By constructing a (possibly large) database of solutions to subproblems, large improvements in search efficiency are possible.

Acknowledgements

Our thanks to Richard Korf for making his implementation of linear conflicts available to us. This work originated out of discussions with Alexander Reinefeld. Thanks to Brent Gorda and the MPCI project at Lawrence Livermore Laboratories for making machine time available to us.

This research was funded by the Natural Sciences and Engineering Research Council of Canada, grants OGP-8173 and OGP-8053.

References

[1] R. Gasser. *Combining Search with Databases: An Improved 15 Puzzle Algorithm*, internal report, Department of Computer Science, ETH Zurich, 1994.

[2] O. Hansson, A. Mayer and M. Yung. Criticizing Solutions to Relaxed Models Yields Powerful Admissible Heuristics. *Information Sciences*, vol. 63, no. 3, pp. 207-227, 1992.

[3] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms* Computer Science Press, 1978.

[4] R. Korf. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, vol. 27, no. 1, pp. 97-109, 1985.

[5] R. Korf. Real-Time Heuristic Search. *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189-211, 1990.

[6] R. Korf. Linear-Space Best-First Search. *Artificial Intelligence*, vol. 62, no. 1, pp. 41-78, 1993.

[7] D. Ratner and M. Warmuth. Finding a Shortest Solution for the $(N \times N)$-Extension of the 15-Puzzle is Intractable, *Journal of Symbolic Computation*, vol. 10, pp. 111-137, 1990.

[8] A. Reinefeld. Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*, *International Joint Conference on Artificial Intelligence*, pp. 248-253, 1993.

[9] A. Reinefeld. private communication, September, 1993.

[10] A. Reinefeld and T. Marsland. Enhanced Iterative-Deepening Search, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, to appear, 1994.

[11] L. Taylor and R. Korf. Pruning Duplicate Nodes in Depth-First Search, *AAAI National Conference*, pp. 756-761, 1993.