Computational Intelligence, Volume –VOL–, Number –NUM–, –YEAR–

PATTERN DATABASES

Joseph C. Culberson and Jonathan Schaeffer

Department of Computing Science, University of Alberta,
Edmonton, Alberta, Canada, T6G 2H1.

The efficiency of A* searching depends on the quality of the lower bound estimates of the solution cost. Pattern databases enumerate all possible subgoals required by any solution, subject to constraints on the subgoal size. Each subgoal in the database provides a tight lower bound on the cost of achieving it. For a given state in the search space, all possible subgoals are looked up, with the maximum cost over all lookups being the lower bound. For sliding tile puzzles, the database enumerates all possible patterns containing N tiles and, for each one, contains a lower bound on the distance to correctly move all N tiles into their correct final location. For the 15-Puzzle, iterative-deepening A* with pattern databases (N=8) reduces the total number of nodes searched on a standard problem set of 100 positions by over 1000-fold.

Key words: Single-agent search, A*, IDA*, pattern databases

Portions of this paper were originally published as: Joseph Culberson and Jonathan Schaeffer, Pattern Databases, *Advances in Artificial Intelligence* (11th Canadian Society for Computational Studies of Intelligence (CSCSI) Conference), Gordon McCalla (ed.), Springer-Verlag, Berlin, pp. 402–416, 1996.

1. INTRODUCTION

The A* search algorithm for single-agent search is of fundamental importance in artificial intelligence. Improvements to the search efficiency can take the range from general algorithm enhancements (application independent) to problem-specific heuristics (application dependent):

- General A* search improvements that are applicable to a wide class of problems. For example, iterative deepening can be used to reduce storage requirements (Korf 1985).
- General search space properties. Many search domains can be represented as directed graphs rather than as trees. The removal of duplicate nodes from the search can result in potentially large savings (Marsland and Reinefeld 1994; Taylor and Korf 1993).
- Branch selection. Given a search tree node with several descendants, search efficiency can be influenced by the order in which the descendants are considered (Marsland and Reinefeld 1994). Although the idea of considering the branch most likely to succeed first is a general principle, the techniques used to make that decision are often application dependent.
- Symmetry reduction. Many problems have inherent symmetries which can be removed. Recognizing and eliminating the symmetries can have an enormous impact on the search space, but it is an application-dependent property.
- Solution databases. In many problems, the states near to the goal nodes can be pre-computed by a backwards search. In two-player games, such as chess and checkers, the backwards searches are saved in endgame databases and are used to stop the search early, improving search efficiency and accuracy (Schaeffer *et al.* 1992). Recently, in single-agent search new bidirectional search methods are creating so-called perimeters around that goal node and using that to improve lower bound cost estimates (Manzini 1995).
- Problem-specific properties. Problem domains may have constraints that can preclude parts of the search space. For example, although the 15-Puzzle has $16! \approx 10^{13}$ positions, only one half of them can be reached from the goal (a parity test can detect this (Horowitz and Sahni 1978)).

Completely general, application-independent search enhancements are of great interest but are rare, while problem-specific tricks are plentiful and usually not of general interest. More often, an idea is presented that has applicability over a class of problems (has partial problem independence) but needs problem-specific information (has partial problem dependence). Often, the more problem-specific the search enhancement, the greater the search savings that are possible.

This paper introduces *pattern databases* as a new approach for enhancing single-agent search. Planning involves deciding on a series of subgoals that are directed towards the solution (Korf 1987). Typically, extensive application-dependent knowledge is required to make the appropriate subgoal choices. Instead, we adopt a "brute force" approach by taking a problem and enumerating all possible subgoals (partial solutions) that satisfy some constraints. At each state in the search, the program can compare its progress towards achieving all possible valid subgoals, and make its next decision based on some or all of this information. We call the set of subgoals a pattern database, because each subgoal is represented as a pattern that can be matched to a search state.

In this paper, pattern databases are applied to permutation problems (although, as shown in Section 6, it generalizes to a wider class of problems). In a *permutation problem*, we have a set of operators that convert one permutation into another. We start with an initial permutation and a specified goal(s). The object is to convert the initial permutation into the goal by applying the operators. In optimization problems, the object is to minimize the number of applications of the operator during the conversion.

The 15-Puzzle, Rubik's Cube, and other similar combinatorial puzzles are examples of permutation problems. A human-like approach to solving such problems non-optimally is to move some of the elements into their correct locations, thereby reducing the complexity of the remaining problem to be solved. This type of "divide-and-conquer" behavior is commonplace in human problem solving. A similar approach is also evident in human play in many multi-player games, for example chess, where the player establishes a plan consisting of a series of intermediate goals. Although the sequence of goals may be non-optimal, the end result is all that matters.

We adapt this idea to the problem of finding optimal solution paths in single-agent search by noting that knowing the minimum distance from a permutation to the nearest other permutation partially matching the goal is a lower bound on reaching the goal. To define a *pattern database* we designate N elements of the goal permutation. Given an arbitrary initial permutation, the locations of the N designated elements form a pattern in the initial permutation. The set of all such patterns forms the domain of the database. Given any permutation, the pattern can be looked up in the pattern database to find the minimal number of operations required to place these N elements in their correct locations. Essentially, a pattern database can be viewed as a collection of solutions to subgoals that must be achieved to solve the problem.

These ideas are illustrated using sliding tile puzzles using iterative-deepening A* (IDA*). This paper uses the 15-Puzzle because the problem is challenging but it is possible to search for optimal solutions. The 24-Puzzle (5×5) has a much larger search space, making it too expensive to search for optimal solutions. Instead, research has concentrated on "good" non-optimal solutions (for example, real-time search (Korf 1990), linear-space best-first search (Korf 1993). 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 (Ratner and Warmuth 1990).

On a standard set of 100 test positions, pattern databases reduce the total number of nodes searched to solve all the problems by 1038-fold compared to using just the Manhattan distance heuristic.

2. PUZZLE SPECIFICATION

The 15-Puzzle is illustrated in Figure 1(a). The puzzle consists of a set of 15 *labeled tiles* arranged in a 4×4 square grid, with one grid location remaining *empty*. We say the empty location holds the *empty tile*. We refer to the tiles by their labels $0 \leq i \leq 15$, where 0 indicates the empty tile. 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.

We label the locations in the puzzle similarly from 0 to 15. A *state*[1] is a permutation of $\langle 0 \ldots 15 \rangle$ *mapping locations to tiles*. The *goal* state of the puzzle is the identity permutation τ as indicated in Figure 1(a).

In any state, a *move* consists of sliding one of the orthogonally adjacent tiles into the empty location. However, we prefer to think of a move as swapping the empty tile with one of its neighbors, and will *specify a move by the direction the empty tile moves*, with l, r, u, d corresponding to *left, right, up* and *down* respectively. Two states are adjacent in the puzzle

[1]We avoid the use of the term "position" since it has two intuitive meanings; 1. the location of a tile and 2. the state of the puzzle.

FIGURE 1. (a) The 15-Puzzle Goal State. (b) State p Induced by $rrdldruu$.

space if one can be obtained from the other in a single move. The *distance* between two states is the minimum number of moves required to produce one from the other. The *cost* of a state is the distance from the goal to the state. A *path* $P(p,q)$ from one state (p) to another (q) is a sequence of moves transforming the first into the second.

The state in Figure 1(b) may be obtained from the goal state by the move sequence $rrdldruu$. In permutation notation, this state is

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 1 & 2 & 0 & 3 & 4 & 9 & 6 & 7 & 8 & 10 & 5 & 11 & 12 & 13 & 14 & 15 \end{pmatrix}.$$

We will use the more compact notation

$$\langle 1\ 2\ 0\ 3\ 4\ 9\ 6\ 7\ 8\ 10\ 5\ 11\ 12\ 13\ 14\ 15 \rangle.$$

As expected, reversing and inverting the move sequence will return to the goal state, in effect inducing the inverse permutation (e.g. $ddlurull$ when applied to the state in Figure 1(b)).

3. PATTERN DATABASES

A *pattern* is the partial specification of a permutation (or state). That is, the tiles occupying certain locations are unspecified. These unspecified tiles are called *blanks*. Note the distinction between blank tiles and the empty tile. In all of the patterns we use in this paper the empty tile will be specified.

A *target pattern* is a partial specification of the goal state. A *pattern database* (PDB) is the set of all patterns which can be obtained by permutations of a target pattern. Two target patterns for which we have built databases are shown in Figure 2. We refer to these databases as the *fringe* and the *corner* respectively.

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

Lemma 1. For any state, for any pattern database, the cost of the pattern induced by the state with respect to the database is a lower bound on the cost of the state.

For example, given an arbitrary 15-Puzzle state, we can map the current locations of tiles 3, 7, 11, 12, 13, 14, 15, and empty into an index into the fringe database. The database

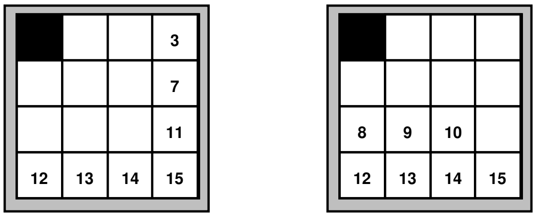

Figure 2. The Fringe and Corner Target Patterns.

will tell us the minimum number of moves required to correctly locate these 8 tiles. This is a lower bound on the remaining search effort. In addition, we could take the current locations for tiles 8, 9, 10, 12, 13, 14, 15, and empty and look this pattern up in the corner database. The value from this database is also a lower bound on the search and may be a better bound (if larger) than that provided by the fringe database.

Other target patterns besides the fringe and corner are possible. Intuitively, these two are likely to be among the best at providing tight lower bounds for the search. To see this, consider the work that remains once the tiles have been moved into the target state. For example, once the fringe pattern has been achieved, all that is left to solve is an 8-Puzzle. Consider a target pattern similar to the fringe, but including tile 10 instead of 15. Having reached the target, some of the work just completed must be undone to correctly place the 15; tiles 10, 11 and 14 are in the way. Our experiments suggest that the best pattern databases are the ones that reduce the problem to a simpler one whose solution requires little or no interference with the placing of the target pattern. Both the fringe and corner patterns satisfy this.

Pattern databases can also be used to determine upper bounds. For example, we computed the fringe database and the "hardest" position requires 61 moves to rearrange the 8 tiles correctly. A global upper bound can be determined by noting that the remaining tiles form an 8-Puzzle. Since the 8-Puzzle has a maximal distance of 31, this gives an upper bound of 92 for any instance of the 15-Puzzle (actually the upper bound is 90 since at least two moves are required to place the empty tile into the upper corner after the other fringe elements are set)[2].

Upper bounds on specific states may be useful in problems where obtaining the optimal solution path is too expensive, for example in larger sliding tile puzzles. For each pattern we may store the permutation associated with one optimal move sequence that takes the current pattern to the target pattern. Then, for any state matching the pattern, applying that permutation results in a new state with only the subpuzzle consisting of the blanks remaining to be solved. On the 15-Puzzle, the subpuzzle for the fringe PDB is just the 8-Puzzle, which can be completely pre-computed and stored. Thus, in just two database lookups we can find an upper bound on any state. For larger problems, we can extend the idea to recursively find pattern databases for the subpuzzles.

If we do not want to store the permutations, then we can use the PDB to select moves on a shortest path to the fringe target pattern. Thus, a (non-optimal) path from initial state

[2]Gasser subsequently used our work to lower the bound to 88 (Gasser 1994). Recent experimental results have lowered this to 82 (Bruenggner *et al.* 1996).

to goal can be obtained in time proportional to the length of the path, giving rapid upper bounds. By broadening the search a little, better upper bounds can be obtained for a little more search cost, essentially using the PDBs as macro-moves.

By modifying the fringe PDB slightly, we can also obtain additional lower bounds and a tighter upper bound. Instead of a single fringe target pattern, we consider the set of nine targets in which the empty tile is located in one of the upper 3×3 locations of the puzzle, and the specified tiles are the same as in the fringe target pattern. We compute the cost of solving each state of the 8-Puzzle by searching it in the 15-Puzzle; that is, we allow the tiles in the fringe targets to be temporarily disturbed in order to solve the 8-Subpuzzle[3]. This gives us a tighter upper bound in some cases.

Using this refined fringe PDB, to obtain a tighter lower bound let f be the cost of reaching a target pattern in this database, and r be the cost of solving the remaining 8-Puzzle. Consider any other path from the current state to the goal, with cost to a fringe pattern 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 $f' + r' \geq r - f$, thus implying a lower bound of $\max\{f, r - f\}$. When $r > 2f$, this gives a better lower bound than f. If it is feasible to wait until the search reaches at least one state in which the target pattern 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.

4. USING SYMMETRY

A technique that can often significantly reduce the search cost is the elimination of subtrees by symmetry. The exact symmetries available will be problem specific, just as the obtaining of lower bounds is. The most commonly seen symmetries are merely reflections of the object under consideration. The symmetries we use for the 15-Puzzle are more subtle, because the set of available moves is dependent on the state. For example, if the empty tile is in a corner we have only two available moves, and for each corner the set of moves is different. Contrast this with Rubik's cube where no matter what state the puzzle is in we have the same set of face rotations available.

Nevertheless, for any square puzzle of side n, we are able to make use of all eight elements of the dihedral group induced by reflections and rotations of the square in the plane[4]. We will let D be the reflection across the main diagonal $(0, 15)$, D' across the transverse diagonal $(12, 3)$, H the horizontal reflection and V the vertical reflection. Rotations can be computed as compositions of these.

We will represent these reflections by the permutations mapping locations induced by the reflections. Thus, for example, the diagonal reflection is the permutation:

$$D = \langle 0\ 4\ 8\ 12\ 1\ 5\ 9\ 13\ 2\ 6\ 10\ 14\ 3\ 7\ 11\ 15 \rangle .$$

We remind the reader that permutations in general, and this group in particular, are associative but not commutative. We also point out that the four reflections are self-inverses. Applying these reflections in all possible compositions gives us a set of 8 isomorphic puzzles, including the original 15-Puzzle (as the identity puzzle). That is, if we apply any of the reflections $R \in \{D, D', H, V\}$ to every state s in the 15-Puzzle, we will generate an isomorphic puzzle. Each R will induce a distinct remapping of the moves, the locations and the tile labels of the 15-Puzzle space.

[3] 172 states of the 8-Puzzle have a reduced cost of 2 when such outside moves are allowed.

[4] For non-square puzzles, fewer transformations would be available.

4.1. The Mirror Symmetry D

We refer to D as the *mirror*. It is the simplest of our mappings to understand, because it has the good fortune to map the 15-Puzzle onto an isomorphic game with the empty cell in the goal state remaining in the upper left hand corner.

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 D induced replacements $l \leftarrow u, u \leftarrow l, r \leftarrow d, d \leftarrow r$ for each move in P. We call P' the *mirror path* of path P. Figure 3 illustrates the path reflection for the path generating the state p in Figure 1(b).

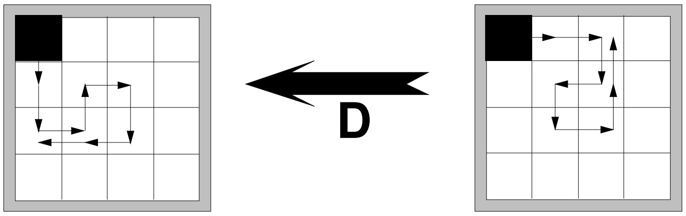

FIGURE 3. Path Reflection.

Figure 4 shows how the mirror states of the puzzle are obtained using D for state p in Figure 1(b). Keeping in mind that compositions are computed from right to left, the

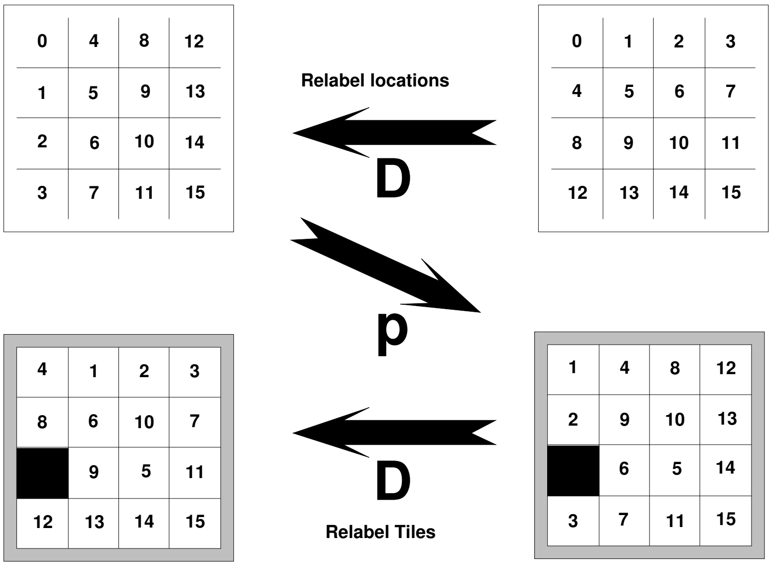

FIGURE 4. Computing the mirror $p' = D \circ p \circ D$.

composition $p \circ D$ means that the locations are first permuted by D as in the top of the figure, then the result is mapped to tiles by p, as indicated in the lower right of Figure 4. If we now apply D (which is equal to D^{-1}) on the left, the effect is to relabel the tiles to correspond to the original puzzle, in the sense that the goal states will be identical. In the lower left of Figure 4 we show the result $p' = D \circ p \circ D$. Undoing the reflected move sequence on the left side of Figure 3 will then yield the goal state of the 15 puzzle.

In short, the mapping $D \circ p \circ D$ applied to all states p in the 15-Puzzle is an automorphism of the puzzle. The proof is straightforward, but the reader is urged to note that it is critical that $D(0) = 0$ in the leftmost multiplication. Otherwise the reflected move sequence, which as noted depends on the location of the empty tile in each state, would not be valid under the relabeling of tiles.

We call the state $p' = D \circ p \circ D$ the *mirror state* of state p. Since for every path $P(\tau, p)$ there is an equal length path $P'(\tau, p')$ and vice versa, it follows that:

Lemma 2. Any upper or lower bound on the cost of a state 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 state. In databases, we need only store information about a state and not its mirror, resulting in a nearly 50% reduction. (We get slightly less than 50% because some states are self-mirrors, for example the goal state). This is a new result, showing that the effective search space for sliding-tile puzzles is half the size previously thought.

4.2. Using the Reflections H,V and D'

If we apply the horizontal reflection ($H = \langle 3\ 2\ 1\ 0\ 7\ 6\ 5\ 4\ 11\ 10\ 9\ 8\ 15\ 14\ 13\ 12\rangle$) to the 15-Puzzle, we obtain an isomorphic puzzle with goal state having tile 0 in the upper right hand corner, as illustrated in Figure 5(a). The induced move reflection swaps l with r, leaving u and d unchanged. However, unlike the D case, we cannot apply H^{-1} on the left to obtain an automorphism of the puzzle, since the relabeling of tile 0 would make the reflected moves invalid.

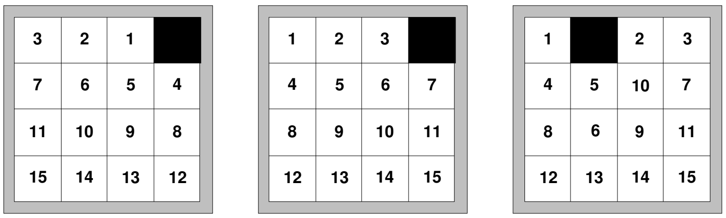

FIGURE 5. (a) Horizontal Goal $\tau^* = \tau \circ H$. (b) Relabeled Goal $\tau'' = \hat{H} \circ \tau \circ H$. (c) $\hat{H} \circ p \circ H$.

Instead, we define $\hat{H} = \langle 0\ 3\ 2\ 1\ 7\ 6\ 5\ 4\ 11\ 10\ 9\ 8\ 15\ 14\ 13\ 12\rangle$ which maps the goal state in Figure 5(a) to the new goal in Figure 5(b). Note that $\hat{H} = \hat{H}^{-1}$. It is straightforward to show that the puzzle $\hat{H} \circ p \circ H, \forall p$, is isomorphic to the standard puzzle.

The usefulness of this mapping lies in the fact that the new goal τ'' requires only 3 moves (lll) to achieve the standard goal τ. Applying the isomorphism, this means that for any state p, if we find lower and upper bounds on the cost of state $p'' = \hat{H} \circ p \circ H$, $L \leq C(p'') \leq U$, then $C(p)$ is bounded by $L - 3 \leq C(p) \leq U + 3$. An example of the lower bound being met is for state $q = \tau$, where $C(q) = 0$ and $C(q'') = 3$. An example of the upper extreme is the state r obtained from τ'' by the move sequence *ddd*. In this case $C(r) = 6$ and $C(r'') = 3$. As a final example, consider state $s = p'' = \hat{H} \circ p \circ H$, the state in Figure 5(c), where p is the position in Figure 1(b). Then $C(s) = 7$, with the solution path *ddrulul*. Reflecting s gives $s'' = \hat{H} \circ s \circ H = p$. The cost of p is $C(s'') = 8$.

The vertical reflection V is analogous to H, with the same difference in bounds applying.

The reflection D' moves the empty tile in the goal to the lower right corner, which requires a minimum of 6 moves to return to the upper right. There are $\binom{6}{3} = 20$ distinct minimal paths, each of which yields a candidate permutation for the relabeling $\widehat{D'}$. We will use this multiplicity of $\widehat{D'}$ in the next section. In any case, letting L and U be lower and upper bounds on $C(p''')$ for $p''' = \widehat{D'} \circ p \circ D'$, then $L - 6 \leq C(p) \leq U + 6$.

Finally, for any state p, we can also compute the mirror state $p' = D \circ p \circ D$, and then apply any of the H, V or D' reflections to obtain further states. Thus, we use all eight symmetries of the square.

4.3. Applying Symmetry to the Databases

Using the properties of mirror states 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 tile is assumed to be blank[5]. Notice, however, that due to the symmetry of the fringe target pattern, the cost of a pattern and its mirror are always equal in the fringe PDB. Using the other symmetries of the 15-Puzzle, even better lower bounds can be obtained. For any state p, additional lower bounds on reaching the goal can be obtained by computing each of the reflected states and looking up the costs in the PDB and subtracting three (for V or H) or six (for D').

For D', the initial reflection carries the empty tile to the lower right corner, requiring six moves to restore the empty tile to the upper right. The 20 distinct paths of length six yield several distinct relabeled tile combinations in the PDB, the exact number depending on the specific PDB in question. Applying each of the resulting $\widehat{D'}$ yields several different lookups for the reflection D'.

We repeat that for each of the states obtained by reflecting across H, V, D', the mirror of the state can also be computed and looked up in the appropriate PDB. None of these mirrors are effective in the fringe because the fringe target pattern 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 pattern is achieved, it is quite unlikely that the specified tiles will be disturbed in completing the rest of the solution. In fact, as previously mentioned, at most two moves can be saved from any state matching the fringe target pattern over the corresponding solution restricted to 8-Puzzle moves. On the other hand, the corner target pattern may be significantly disturbed in order for the solution to be completed. This argument places some intuitive limits on which target patterns are likely to be useful. For example, a target pattern in which only alternate tiles are specified could seemingly be very far from the goal. Such a database would likely yield weak lower bounds for most states.

[5] Our program, for reasons of efficiency, does not compute the images of blank tiles, only those actually in the PDB.

4.4. Applying Symmetry to Search

So far, our program uses only the mirror states to prune the search tree itself. We only use the other isomorphisms to obtain additional results from the database, as discussed above.

It is also possible to use the mirror transformation in a backwards search, from goal to the start state. Because of mirrors, there are (usually) two start states, but only one goal state since the goal is a self-mirror. Holst has shown significant savings using this technique (Holst 1995). Fewer nodes are searched, although the cost of a search node has increased.

5. EXAMPLE

Table 1 shows the results of looking up a position in the fringe database (position number 79 in the Korf set (Korf 1985), the simplest problem). The table shows all the images for the original position (O): horizontal (OH), vertical (OV), and 6 diagonals (OD). The mirror (M) is also shown for illustrative purposes. Permutations of the mirror (MH, MV and MD) are not shown since they do not improve the quality of the solution (because of the fringe's symmetry). The following code shows how the table entry was generated for the horizontal permutation:

for(i = 0; i <= 15; i++)
    OH[H[i]] = $\hat{H}$[O[i]];

Note that the values in the original position are remapped, and only those values that form part of the fringe (0, 3, 7, 11, 12, 13, 14, 15) are relevant for interrogating the database. All other values are irrelevant, and are shown as dashes in the table.

Name	Mapped Position	PDB	Penalty	Bound
O	0 1 9 7 11 13 5 3 14 12 4 2 8 6 10 15	38	0	38
OH	- - 3 0 - - 14 - - 7 15 13 12 - - 11	29	-3	26
OV	- - - 3 - - 12 14 7 - - 15 0 13 - 11	31	-3	28
OD	- - - 7 14 11 3 - 13 - - - 12 - 15 0	34	-6	28
OD	- - 13 7 14 11 3 - 12 - - - - - 15 0	38	-6	32
OD	- - - 7 13 11 3 - 12 14 - - - - 15 0	30	-6	24
OD	- - - 11 13 15 7 - 12 - 3 - - - 14 0	38	-6	32
OD	- - - 11 13 15 3 - 12 - - - - 7 14 0	36	-6	30
OD	- - - 7 13 15 3 - 12 11 - - - - 14 0	34	-6	28
M	0 14 11 - - 7 3 - - - - - 13 12 - 15	38	0	38

TABLE 1. Position 79 Example.

The most commonly used lower-bound estimator for sliding tile puzzles is the Manhattan distance. 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). Table 2 shows the results of looking up all 100 Korf test positions in the fringe and corner databases. The fringe results in an average improved lower bound of 6.34 over using Manhattan distance, slightly better than the corner's value of 5.72. If both databases were used, the average improvement would be 7.08.

Pattern Database	Improvement over Manhattan Distance +14	+12	+10	+8	+6	+4	+2	0	Average
Fringe	2	4	16	22	24	15	9	8	6.34
Corner	2	4	10	13	32	17	16	6	5.72
Combined	3	8	16	24	25	13	8	3	7.08

TABLE 2. Benefits of Pattern Databases.

6. EXPERIMENTS

Iterative-deepening A* for the 15-Puzzle was implemented using the Manhattan distance heuristic estimate (MD), fringe pattern database (FR) and corner pattern 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[6]. The programs were written in C and run on a BBN TC2000 at Lawrence Livermore National Laboratory. Each database took less than one hour of real time to compute. 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 (Korf 1985).

Table 3 shows that the fringe database (FR) reduces the search trees by 346-fold, and the corner database (CO) is even better with a 437-fold reduction. Since the fringe is symmetric, it has fewer patterns to look up (9: O, OH, OV and 6 OD) than for the corner (26: O, OH, OV, 10 OD, M, MH, MV, 10 MD). Thus, the corner's lower bound is maximized over more values, generally giving a tighter bound. Detailed results can be found in Appendix A.

Experiment	Total Nodes	Tree Size %	Improvement
MD	36,302,808,031	100.00	1
MD+FR	105,067,478	0.29	346
MD+CO	83,125,633	0.23	437
MD+FC	34,987,894	0.10	1038

TABLE 3. Results Summary.

Examination of the data produced by the pattern databases showed that for many positions they produced complimentary results: positions where the corner did poorly often

[6]Because of the symmetry of the fringe, it requires only half the storage.

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 pattern database was broken into 16 parts - one for each square that the blank could be on. 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 6a). Given a position, the patterns for which the appropriate pattern database is in memory are looked up. For example, Figure 6b shows the position of an empty square (O) and where it would appear in all the patterns of that position. The original position (O), its mirror (M), corner diagonal (DO) and corner diagonal mirror (DM) patterns 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) patterns would be looked up in the fringe database. When the empty square moves once, the original position, mirror, fringe DO and fringe DM patterns would be looked up using the fringe database, while the corner VO, the corner VM, the HO and the corner HM patterns 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.

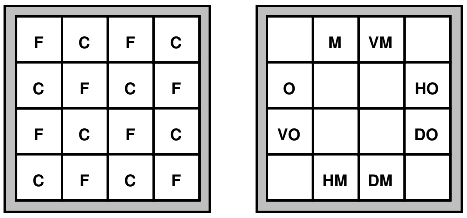

FIGURE 6. a) 15-Puzzle as a Checkerboard. b) Looking up a Position.

Using half of the corner and fringe databases (FC) without any other enhancements results in the tree size being reduced by 1038-fold. Using FC gives an almost 2.5-fold improvement over just using the corner database. In other words, the union of the two databases (FC) is better than the sum of using them individually. Detailed results can be found in the Appendix A.

We also experimented with using upper bounds. IDA* starts with a lower bound on the optimal answer and repeatedly iterates, increasing the bound, until the answer is reached. Upper bounds, if effective, can therefore only prevent the last iteration from having to be being done. If the upper bound is too high, then there are no savings. If the upper bound is correct, then the last iteration can be eliminated since the lower bound will have been proven to be equal to the upper bound. In our experiments, this condition only occurred in 3 of the 100 problems, and did not significantly influence the final performance numbers. Although upper bounds were not effective in our application, they still may be useful as estimates of the amount of search effort remaining.

The linear conflicts heuristic (LC) is a recently proposed improvement to the Manhattan distance heuristic (Hansson, Meyer and Yung 1992). 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). Linear conflicts alone reduce the Manhattan trees by 9.7-fold. Using pattern databases, our best LC results were only 7% better than our best MD results, implying most of the benefits of linear conflicts are captured in the pattern databases.

In Table 4, the 100 positions have been sorted into groups of 20 based on the number of nodes required to solve the MD tree. We used two metrics to compare performance: AVG is the average of the individual search reductions computed as $(\sum_{i=1}^{N} MD_i/PDB_i)/N$ and TTL is the ratio of the total tree sizes computed as $\sum_{i=1}^{N} MD_i/\sum_{i=1}^{N} PDB_i$. The results show that the 20 easiest problems benefit by a factor of 240 TTL (521 for AVG) using databases, while the 20 hardest problems benefit by a factor of 1,179 (2,499 for AVG). This suggests that the improvements are limited only by the small problems, whose solution trees become close to the minimal search tree.

Problems	AVG	TTL
01-20	521	240
21-40	581	315
41-60	1360	476
61-80	1502	737
81-100	2499	1179
Overall	1293	1038

TABLE 4. 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 patterns at each node (as many as 26), rather than maintaining all the information incrementally. Further, the program uses the maximum number of patterns for computing a bound. The diagonal patterns 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 FC by itself, with only the original and mirror position queried in the pattern 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. In addition, other search enhancements (such as transposition tables or solution databases) can be used to further reduce tree size, again affecting the tree size / execution time ratio. Our experiments show that further reductions can be achieved; using transposition tables and solution databases with pattern databases resulted in an average 1775-fold reduction in tree size for the 15-Puzzle (Culberson and Schaeffer 1994).

Our experiments used the entire fringe and corner databases to maximize the amount of information available to the search. Clearly, space/time trade-offs are possible. Gasser has subsequently taken part of our fringe database (the 4 central squares) and built a 15-Puzzle searcher that searches efficiently with less memory/disk requirements (Gasser 1994).

7. FURTHER RESEARCH

Pattern 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 pattern databases at run-time will improve.

Pattern databases are more expensive to compute than other lower bounds, but they are pre-computed in a pre-search phase. For the puzzle environments considered, the amortized cost over all possible searches is negligible. (This is also the case for solution databases.) For other problems such as the Traveling Salesman, the application of pattern databases will be limited since each problem is from a different space. One can imagine, however, that given a map of North America, there might be a need for being able to supply quickly good approximations to tours of a subset of the cities. In such an environment it may be possible to use pre-computed databases of partial results to more quickly obtain good upper bound approximations.

This work can be extended to the 24-Puzzle. For the 15-Puzzle, the pattern 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 (Korf 1990). The database information can be used to increase the accuracy of any heuristic estimate. The larger the puzzle, the more effective become the patterns. 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 pattern database of the patterns 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 pattern database maximum value, search the one with the smallest sum (or average) of the individual database pattern 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 instance 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 pattern 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, improvements in search efficiency are possible.

To illustrate its generality, this work has been applied to the checkers-playing program *Chinook* (Schaeffer *et al.* 1992). In the alpha-beta search algorithm, the search should only stop in so-called quiescent (quiet) positions. In checkers, the most common tactical combination is a "2 for 1": allow a man to be captured to get a double capture in return. An analysis of search trees showed that 50% of these combinations occurred in a specific region of the board. A pattern database was built that enumerated all the piece/square placings that could be part of this type of combination. For each legal placing, an off-line search was done to verify that the combination worked. During a game, a potential leaf node position is mapped into a pattern and checked to see if the tactical combination is present. If so, the search is extended to see the consequences. The program *knows* it is winning a checker; it must search a bit deeper to see if the opponent has any resources after the checker is lost (for example, has a return "2 for 1"). With the sliding tile puzzles, pattern databases were used to curtail search; here they are used to extend search. The addition of this database has been a major addition to *Chinook*.

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. Aske Plaat provided us with useful feedback on the text.

This research was funded by the Natural Sciences and Engineering Research Council of Canada, grants OGP-8173 and OGP-8053, and the Netherlands Organization for Scientific Research (NWO).

REFERENCES

CULBERSON, J. and J. SCHAEFFER. 1994. Efficiently Searching the 15-Puzzle, TR 94-08, Department of Computing Science, University of Alberta.

GASSER, R. 1994. Harnessing Computational Resources for Efficient Exhaustive Search. Ph.D., ETH Zurich, Switzerland, December.

BRUENGGER, A., A. MARZETTA, K. FUKUDA and J. NIEVERGELT. 1996. The Parallel Exhaustive Search Workbench ZRAM and its Applications. Proceedings of the Parallel Optimization Colloquium, Versailles, France.

HANSSON, O., A. MAYER and M. YUNG. 1992. Criticizing Solutions to Relaxed Models Yields Powerful Admissible Heuristics. *Information Sciences*, vol. 63, no. 3, pp. 207–227.

HOLST, W. 1995. Unpublished research, University of Alberta.

HOROWITZ, E. and S. SAHNI. 1978. *Fundamentals of Computer Algorithms,* Computer Science Press.

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

KORF, R. 1987. Planning as Search: A Quantitative Approach *Artificial Intelligence*, vol. 33, no. 1, pp. 65–88.

KORF, R. 1990. Real-Time Heuristic Search. *Artificial Intelligence*, vol. 42, no. 2–3, pp. 189–211.

KORF, R. 1993. Linear-Space Best-First Search. *Artificial Intelligence*, vol. 62, no. 1, pp. 41–78.

MANZINI, G. 1995. BIDA*: An Improved Perimeter Search Algorithm *Artificial Intelligence*, vol. 75, no. 2, pp. 347–360.

RATNER, D. and M. WARMUTH. 1990. 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.

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

REINEFELD, A. 1993. Private communication, September.

REINEFELD, A. and T. MARSLAND. 1994. Enhanced Iterative-Deepening Search, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 7, pp. 701–710.

SCHAEFFER, J., J. CULBERSON, N. TRELOAR, B. KNIGHT, P. LU and D. SZAFRON. 1992. A World Championship Caliber Checkers Program, *Artificial Intelligence*, vol. 53, no. 2–3, pp. 273–290.

TAYLOR, L. and R. KORF. 1993. Pruning Duplicate Nodes in Depth-First Search, *AAAI National Conference*, pp. 756–761.

APPENDIX A.

This appendix compares the tree size when using the Manhattan Distance (MD) in conjunction with the fringe pattern database (FR), corner pattern database (CO), and using both the fringe and corner on alternate squares (FC).

#	MD	MD+FR	%	MD+CO	%	MD+FC	%
1	276361933	50773	0.02	306806	0.11	36243	0.01
2	15300442	395581	2.59	106795	0.70	104602	0.68
3	565994203	1928333	0.34	4380373	0.77	1136069	0.20
4	62643179	78073	0.12	199322	0.32	52596	0.08
5	11020325	407978	3.70	246838	2.24	187116	1.70
6	32201660	178027	0.55	150849	0.47	65161	0.20
7	387138094	1115121	0.29	1971150	0.51	754182	0.19
8	39118937	125578	0.32	117455	0.30	70313	0.18
9	1650696	10982	0.67	8010	0.49	5376	0.33
10	198758703	3041796	1.53	1266145	0.64	919680	0.46
11	150346072	898237	0.60	784647	0.52	405740	0.27
12	546344	9302	1.70	6562	1.20	3623	0.66
13	11861705	27251	0.23	29470	0.25	8284	0.07
14	1369596778	236249	0.02	185530	0.01	96880	0.01
15	543598067	1483214	0.27	2723255	0.50	534876	0.10
16	17984051	101034	0.56	19760	0.11	21477	0.12
17	607399560	1150412	0.19	2027647	0.33	613740	0.10
18	23711067	19750	0.08	106787	0.45	11095	0.05
19	1280495	17566	1.37	13353	1.04	6579	0.51
20	17954870	182273	1.02	165988	0.92	102741	0.57
21	257064810	655745	0.26	1009255	0.39	371882	0.14
22	750745755	2759305	0.37	1544033	0.21	748119	0.10
23	15971319	133065	0.83	42319	0.26	30445	0.19
24	42693209	998053	2.34	84271	0.20	94728	0.22
25	100734844	135828	0.13	358614	0.36	88543	0.09
26	226668645	911934	0.40	820333	0.36	450264	0.20
27	306123421	546652	0.18	146288	0.05	75511	0.02
28	5934442	3972	0.07	38331	0.65	3345	0.06
29	117076111	148946	0.13	166939	0.14	85889	0.07
30	2196593	3859	0.18	9262	0.42	3031	0.14
31	2351811	1252	0.05	10336	0.44	1219	0.05
32	661041936	10694675	1.62	4344085	0.66	3966538	0.60
33	480637867	1432634	0.30	1011703	0.21	496454	0.10
34	20671552	210345	1.02	148608	0.72	77218	0.37
35	47506056	37026	0.08	12315	0.03	5028	0.01
36	59802602	154986	0.26	164233	0.27	71238	0.12
37	280078791	393629	0.14	1363809	0.49	290466	0.10
38	24492852	53780	0.22	225209	0.92	38011	0.16
39	19355806	377262	1.95	28001	0.14	36323	0.19
40	63276188	268950	0.43	260678	0.41	129719	0.21
41	51501544	854252	1.66	197439	0.38	133814	0.26
42	877823	2520	0.29	13190	1.50	2563	0.29
43	41124767	294807	0.72	1528992	3.72	242465	0.59
44	95733125	129796	0.14	114815	0.12	75611	0.08
45	6158733	28222	0.46	49786	0.81	20047	0.33
46	22119320	72910	0.33	76328	0.35	31892	0.14
47	1411294	4474	0.32	1339	0.09	963	0.07
48	1905023	73887	3.88	23701	1.24	20010	1.05
49	1809933698	1591321	0.09	1668463	0.09	760504	0.04
50	63036422	490882	0.78	323861	0.51	246627	0.39

Table 5. Results for the 100 Korf Positions (I).

#	MD	MD+FR	%	MD+CO	%	MD+FC	%
51	26622863	221716	0.83	182202	0.68	102211	0.38
52	377141881	1879450	0.50	408995	0.11	307787	0.08
53	465225698	2832184	0.61	921751	0.20	476433	0.10
54	220374385	322076	0.15	261314	0.12	156892	0.07
55	927212	4983	0.54	2299	0.25	2012	0.22
56	1199487996	1197142	0.10	770455	0.06	478569	0.04
57	8841527	15255	0.17	51285	0.58	12098	0.14
58	12955404	249942	1.93	31142	0.24	30184	0.23
59	1207520464	1267614	0.10	248605	0.02	214422	0.02
60	3337690331	25938931	0.78	10464802	0.31	6086900	0.18
61	7096850	26086	0.37	49094	0.69	16820	0.24
62	23540413	81056	0.34	223550	0.95	40979	0.17
63	995472712	4133859	0.42	1004241	0.10	779656	0.08
64	260054152	329876	0.13	688083	0.26	249683	0.10
65	18997681	58339	0.31	54227	0.29	26226	0.14
66	1957191378	1710785	0.09	10487203	0.54	1367159	0.07
67	252783878	875080	0.35	180294	0.07	164340	0.07
68	64367799	92981	0.14	127960	0.20	30420	0.05
69	109562359	115228	0.11	46234	0.04	33540	0.03
70	151042571	1041280	0.69	202868	0.13	192382	0.13
71	8885972	6846	0.08	10697	0.12	4435	0.05
72	1031641140	147908	0.01	666232	0.06	128658	0.01
73	3222276	49076	1.52	15688	0.49	13507	0.42
74	1897728	88116	4.64	43205	2.28	35911	1.89
75	42772589	620859	1.45	86401	0.20	79080	0.18
76	126638417	1134130	0.90	456423	0.36	293955	0.23
77	18918269	249209	1.32	144519	0.76	85067	0.45
78	10907150	261047	2.39	44874	0.41	43256	0.40
79	540860	3735	0.69	5303	0.98	2367	0.44
80	132945856	1646046	1.24	851437	0.64	624092	0.47
81	9982569	21073	0.21	25176	0.25	12974	0.13
82	5506801123	2824992	0.05	3375958	0.06	1207363	0.02
83	65533432	39329	0.06	33115	0.05	18707	0.03
84	106074303	171367	0.16	240396	0.23	90085	0.08
85	2725456	9865	0.36	5816	0.21	4246	0.16
86	2304426	51492	2.23	20750	0.90	18547	0.80
87	64926494	43384	0.07	150305	0.23	36177	0.06
88	6320047980	12046043	0.19	15206418	0.24	4631654	0.07
89	166571021	1665455	1.00	527385	0.32	480785	0.29
90	7171137	55353	0.77	15858	0.22	12142	0.17
91	602886858	1667211	0.28	1871184	0.31	891789	0.15
92	1101072541	2424245	0.22	1086972	0.10	834259	0.08
93	1599909	5027	0.31	5999	0.37	2634	0.16
94	1337340	56874	4.25	48559	3.63	30078	2.25
95	7115967	141359	1.99	25225	0.35	27227	0.38
96	12808564	147710	1.15	29885	0.23	33187	0.26
97	1002927	9272	0.92	3235	0.32	3209	0.32
98	183526883	435601	0.24	161259	0.09	108344	0.06
99	83477694	435365	0.52	264761	0.32	154128	0.18
100	67880056	1293127	1.91	692716	1.02	576478	0.85
Ttl	36302808031	105067478	0.29	83125633	0.23	34987894	0.10
Ttl	Reduction		346		436		1038
Avg	Reduction		581		606		1293

TABLE 6. Results for the 100 Korf Positions (II).