The Manhattan Pair Distance Heuristic for the 15-Puzzle

Technical Report PC^2 /TR-001-94

PC^2

PADERBORN
CENTER FOR
PARALLEL
COMPUTING

Bernard Bauer, PC^2 – Universität-GH Paderborn
e-mail: bb@uni-paderborn.de

33095 Paderborn, Warburger Str. 100

January 14, 1994

The Manhattan Pair Distance Heuristic for the 15-Puzzle

Bernard Bauer
University of Paderborn
Paderborn Center for Parallel Computing
33095 Paderborn, Germany
phone: +49 5251 60 3334, email: bb@uni-paderborn.de

Abstract

The 15-puzzle is a popular workbench model for measuring the performance of heuristic search algorithms. While much effort has been spent on improving the search algorithms, less attention has been paid to derive powerful heuristic estimate functions which guide the search process into the most promising parts of the search tree.
In this paper, we present the Manhattan Pair Distance (MPD), a combination of the well-known Manhattan Distance function and our new Pair Distance heuristic. Using Iterative Deepening A* as the basic search procedure, we achieved an average node saving of more than 80 % on Korf's standard set of 100 random puzzle instances. Due to the increased book-keeping overheads, these node count savings correspond to a 50 % reduction of the total run-time, when using a highly efficient recursive implementation of Korf's IDA*.

Keywords

Heuristic search, 15-puzzle, Iterative-Deepening A*

1 Introduction

Sam Loyd's 15-puzzle is one of the best representatives for a type of games keeping many people busy for a long time trying to find a shortest (= optimal) solution. Although already invented in 1878 [2] it is still a favorite model for measuring the performance of heuristic search algorithms in (super-) computers.

Fifteen squared tiles numbered from 1 to 15 are located in a square tray of size 4×4 and have to be re-arranged to a given goal configuration. Because of the single blank square identified by the *blank tile* an orthogonally adjacent tile can slide into this position. This *move* changes the positions of the two involved tiles. The extension of the puzzle to a $n \times k$ puzzle is obvious. In this more general formulation, the problem to find an optimal solution is `NP-hard` [7]. Indeed, the decision problem whether there exists a sequence of moves that changes a given start configuration to a goal state within less than k moves is `NP-complete`. This is a convincing argument for the difficulty of the problem.

Despite its complexity, we look at the problem to find *optimal* solutions. Because of the simple rules, the easy implementation, the large search space and the sparse solution density it is very suitable for analyzing the behavior of search algorithms. For that reasons it is of interest itself, too [8, 9].

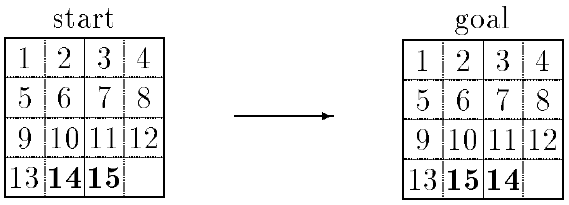

Figure 1: The original (unsolvable) puzzle problem

Looking at the problem from an algebraic point of view a configuration can be identified as an element (= permutation) of the symmetrical group S_{n^2}. Then a moves corresponds to a transposition. With the properties of even permutations and the sign of a permutation it can be shown that only half of the $(n^2)!$ possible configurations can be reached from a given start state - thus leaving the original problem in Fig. 1 unsolvable[1] [10]. However, with the remaining $16!/2 \approx 10^{13}$ configurations the problem space is still of non-trivial size.

One of the reasons for using the puzzle as a benchmark is its easy implementation. Moves can be generated very quickly and the goal state is checked in constant time. The principal difficulty in solving the problem lies in the size of the search space. Uninformed *brute-force* searches like depth-first or breadth-first searches (DFS, BFS rsp.) cannot be applied. Certainly, DFS handles the limited storage size but is not guaranteed to find a solution at all. BFS will find an optimal solution if it does not overflow the restricted storage size already before [5].

One common method to overcome these restrictions is the usage of a heuristic function $\tilde{h}$ in those algorithms. $\tilde{h}$ is an estimation of the *cost* (= moves) to reach the given goal from the current state in the search. Combining $\tilde{h}$ with an estimation $\tilde{g}$ (the costs already accumulated on the way passed) to $\tilde{f} = \tilde{g} + \tilde{h}$, the search can be guided to the goal direction. Optimality is guaranteed if $\tilde{g}$ does not underestimate and $\tilde{h}$ does not overestimate the solution cost. The resulting search algorithm is then called *admissible* [6].

A well-known heuristic *best-first* search is the A^* algorithm [6, 5]. At every search step, A^* expands the best node (the one with lowest cost bound $\tilde{f}$) storing its successors with updated cost values onto a stack. Again, this leads to a very fast growing stack, similar to the uninformed BFS.

Combining the advantages of the above algorithms leads to IDA^* [3, 4]. First, the combination of DFS and BFS leads to $DFID$ (depth-first iterative deepening) which conducts a series of inde-

[1]... which does not prevent many human players to attack this problem, anyway

pendent (backtracking) searches, each with a look-ahead horizon extended by an additional tree level. In IDA^* the successive iterations do not correspond to increased search depths, but to increased cost bounds of the currently investigated solution space. This reduces the space complexity to linear while preserving the optimality of the solution.

As a consequence, IDA^* can be applied to our puzzle problem. Even so, problem instances greater than 4×4 cannot be solved in reasonable time. In contrast to many other combinatorial problems there exists only one good and commonly used heuristic function, the *Manhattan Distance* or *city-block distance.* This estimate is the sum of the minimum moves of each tile ($\neq$ blank tile) from its current position to the goal position. Obviously MD is admissible, because it disregards any blocking tiles between the current and the final position of the moving tile.

The rest of this paper is organized as follows: After some necessary definitions we will have a closer look at the MD and the new Pair Distance. Some results from practical tests with the 15-puzzle are shown next. A concluding section summarizes the results and gives ideas for further work.

2 The Graph Search Problem

The best way to look at the puzzle problem is its definition as a graph search problem[2].

Definition 1 (Configuration, State):
A *configuration* or *state* of a $n \times n$ puzzle (tray), $n \geq 2$, is a placement of the $1, ..., n^2 \Leftrightarrow 1$ tiles in the n^2 squares of the $n \times n$ tray. Exactly one tile is placed on one square - thus leaving one square empty. This square is occupied by the *blank tile*, tile no. 0. The squares can be numbered consecutively from $\{0, ..., n^2 \Leftrightarrow 1\}$ or in the

[2]Without loss of generality we investigate only quadratic trays

way matrices are numbered. A tile x is identified by its number or its position (i_x, j_x), the i-th square in the row and the j-th square in the column, $i, j \in \{0, ..., n \Leftrightarrow 1\}$.

Definition 2 (Puzzle Graph):
Let $\mathcal{V}$ be the set of configurations of a $n \times n$-puzzle and $\alpha, \beta \in \mathcal{V}$ with $\alpha \neq \beta$ be two distinct configurations. A *regular operation or move* exchanges the blank tile with an orthogonally adjacent tile and transfers α into β. The equivalence relation $\sim$ is defined as: $\alpha \sim \beta :\Leftrightarrow \exists$ a regular operation for $\alpha, \beta \in \mathcal{V}$. $\sim^*$ then denotes the transitive closure.

$PG_n = (\mathcal{V}, \mathcal{E})$ denotes the undirected puzzle graph with
$\mathcal{V}$ = the set of configurations of a $n \times n$-puzzle
$\mathcal{E} = \{(\alpha, \beta) \mid \alpha, \beta \in \mathcal{V} \wedge \alpha \sim^* \beta\}$

Definition 3 (Manhattan Distance):
Let α, β be two (solvable) $n \times n$-puzzle configurations and (i_x^α, j_x^α) denote the position of tile x in configuration α ((i_x^β, j_x^β) for β rsp.). The *Manhattan Distance* is defined:

$$MD(\alpha, \beta) = \sum_{x=1}^{n^2-1} (|i_x^\alpha \Leftrightarrow i_x^\beta| + |j_x^\alpha \Leftrightarrow j_x^\beta|)$$

Note: The blank tile (no. 0) is not included in the sum. If not stated otherwise we will think of the goal state as shown in Fig. 3c on page 7, with tile number i placed on square i in the tray.

3 The Manhattan Pair Distance

Generally, an improved heuristic function should be more accurate than the Manhattan Distance - thereby generating fewer nodes during the search - and it should also decrease the total search time. The Manhattan Distance can be calculated efficiently by updating the heuristic costs from a pre-calculated three-dimensional table (in the size of the puzzle instance n).

Our *Pair Distance* is a little bit more complex. The idea derives from the original puzzle problem of Sam Loyd, shown in Fig. 1 [1]. Here, the adjacent tiles 14 and 15 are exactly changed on their goal positions. The Manhattan Distance function returns the value 2 as a minimum value of moves in order to change the tiles. In fact, such a transposition can never be performed in only 2 moves. Instead one needs at least two further moves.

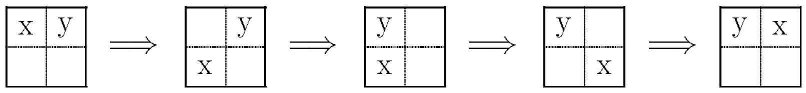

Figure 2: tiles x and y changing their position

In an arbitrary state two orthogonally adjacent tiles ($\neq$ blank) can never change their positions in only two moves as indicated by the MD. Instead, at least 4 moves are required for the exchange: One tile has to leave its location to another location which is different from its goal position. Then, the other tile can be moved directly to its goal square. But there remains a minimum distance of 2 for the first tile on its way to the final location. Fig. 2 shows this procedure, disregarding any blocking tiles. Clearly, the Pair Distance does not overestimate the number of moves which have to be done and is admissible (an important fact for optimally).

This idea can simply be generalized to a more efficient heuristic, the 'real' Pair Distance. Consider two tiles x, y that are already placed on any square in the goal row but in the wrong order. Like two adjacent tiles at least one of them has to leave the *final row* for interchanging which leads us to the same result as for directly adjacent tiles. The same consideration holds for columns.

The examples of Fig. 3 elucidate the Pair Distance. Fig. 3a shows an interesting case with several pairs. First, the tiles 11 and 3 build a pair in the last column, 10 and 8 in the third row. The pairs 5-4 and 7-6 are placed in the same row. The same holds for the pairs 14-15 and 12-13, too. Interestingly, in this fourth row 14-12 and 15-13 form pairs as well.

	1	2	11
5	4	7	6
10	9	8	3
14	15	12	13

a: MD = 20
PD = 12

9	2	3	1
14	4	5	2
8	11	10	
12	13	6	15

b: MD = 19
PD = 6

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

c: common goal

Figure 3: some example configurations

Two further remarkable cases are shown in Fig. 3b. Tile 10 is related to two pairs at the same time, with tiles 11 in the same row and tile 6 in the same column. But, as can be seen for tile 1 and the possible partner tiles 2 and 3, a tile can build at most one pair relation in the same row, column rsp.. If tile 1 changes the row both tiles 2 and 3 can slide to the right. Her, counting PD for two pairs the required number of moves.

The formal definition of the PD is based on two functions: Row and Col. For a given tile x Row returns the set of suitable pair partners in the same row. Col works with respect to the column accordingly. We use the functions in our algorithmically description of the PD.

Let $\alpha, \beta \in \mathcal{V}$ be two configurations and x be any tile with $x = (i_x^\alpha, j_x^\alpha)$ in α and $x = (i_x^\beta, j_x^\beta)$ in β, in short $x^{\alpha,\beta}$.

$$\textsc{Row}(x^{\alpha,\beta}) = \left\{ \begin{array}{lll} y \mid & & \text{tile } y \neq 0,\ y = (k_y^\alpha, l_y^\alpha), \text{ rsp. } (k_y^\beta, l_y^\beta) \\ & & k, l \in \{0, \ldots, n-1\} : \\ & & ((i_x^\alpha = k_y^\alpha = i_x^\beta = k_y^\beta) \wedge (j_x^\alpha < l_y^\alpha) \wedge (j_x^\beta > l_y^\beta)) \\ & \vee & ((i_x^\alpha = k_y^\alpha = i_x^\beta = k_y^\beta) \wedge (j_x^\alpha > l_y^\alpha) \wedge (j_x^\beta < l_y^\beta)) \end{array} \right\}$$

$$\textsc{Col}(x^{\alpha,\beta}) = \left\{ \begin{array}{lll} y \mid & & \text{tile } y \neq 0,\ y = (k_y^\alpha, l_y^\alpha), \text{ rsp. } (k_y^\beta, l_y^\beta) \\ & & k, l \in \{0, \ldots, n-1\} : \\ & & ((j_x^\alpha = l_y^\alpha = j_x^\beta = l_y^\beta) \wedge (i_x^\alpha < k_y^\alpha) \wedge (i_x^\beta > k_y^\beta)) \\ & \vee & ((j_x^\alpha = l_y^\alpha = j_x^\beta = l_y^\beta) \wedge (i_x^\alpha > k_y^\alpha) \wedge (i_x^\beta < k_y^\beta)) \end{array} \right\}$$

Because a tile x cannot have more than one partner in the same row (column, rsp.), tiles already involved in a row pair relation are stored in the row-list. The column-list contains tiles in a column pair relation.

Definition 4 (Pair Distance):
Let α, β be two (solvable) $n \times n$-puzzle configurations and (i_x^α, j_x^α) denote the position of tile x in configuration α ((i_x^β, j_x^β) for β rsp.). The tiles are numbered from 0 (the blank tile) to $n^2 \Leftrightarrow 1$. The following algorithm computes $PD(\alpha, \beta)$:

Input: α, β configurations
Output: the Pair Distance of α with respect to β

```
algorithm $PD(\alpha,\beta)$;
pairs := 0;
set row-list := $\emptyset$; column-list := $\emptyset$;
for ($x = 1$ to $n^2 - 1$) do {
    if $x$ not in row-list
        if ( ( $\exists\, y \in \textsc{Row}(x^{\alpha,\beta})$ ) $\wedge$ ( $y$ not in row-list ) )
            insert $x$ and $y$ into the row-list;
            pairs := pairs + 2;
    if $x$ not in column-list
        if ( ( $\exists\, y \in \textsc{Col}(x^{\alpha,\beta})$ ) $\wedge$ ( $y$ not in col-list ) )
            insert $x$ and $y$ into the col-list;
            pairs := pairs + 2;
}
return ( pairs )
```

Only for the moving tile the *for*-loop is applied during the search, thus leading to linear running time for the PD algorithm. In general, the complexity is $O(n^3)$. From the separation of rows and columns it follows that a tile can build at most two pairs at the same time, one in the row and one in the column. For reasons of simplicity we do not consider multiple pair relations of one tile in our tests.

Of course, PD is not useful as a stand-alone heuristic function. But in conjunction with the MD it can be combined to an efficient heuristic which saves a great amount of node expansions while operating at low costs. Obviously, MPD is still admissible.

Definition 5 (Manhattan Pair Distance):
Let α, β be two (solvable) $n \times n$-puzzle configurations. The Manhattan Pair Distance is:

$$MPD(\alpha, \beta) = MD(\alpha, \beta) + PD(\alpha, \beta)$$

4 Experimental Results

To confirm our theoretical results we tested the MPD heuristic with Korf's IDA* algorithm and his well-known set of 100 problem instances [3]. We ran all instances once with MD and once with MPD.

We made only three changes to our IDA* source code: We introduced a global array of size $n \times n$ used to store tiles which are involved in a pair relation, the functions *remove-pairs* for dissolving tiles' partnership when one tile (as a partner) leaves the pair association and *set-pairs* setting a new partnership after a move for a tile.

Thereby we did not search a new partner for the remaining tile when we call remove-pairs. In fact, this remaining tile could find a new partner but this causes much overhead in the recursion and we found it not useful for the small 15-puzzle.

We distinguish two cases: finding the first solution and finding all solutions. The *all-solutions-case* is useful to find out about possible irregularities of the algorithm and/or problem domain (e.g. solution density and quantity) [8].

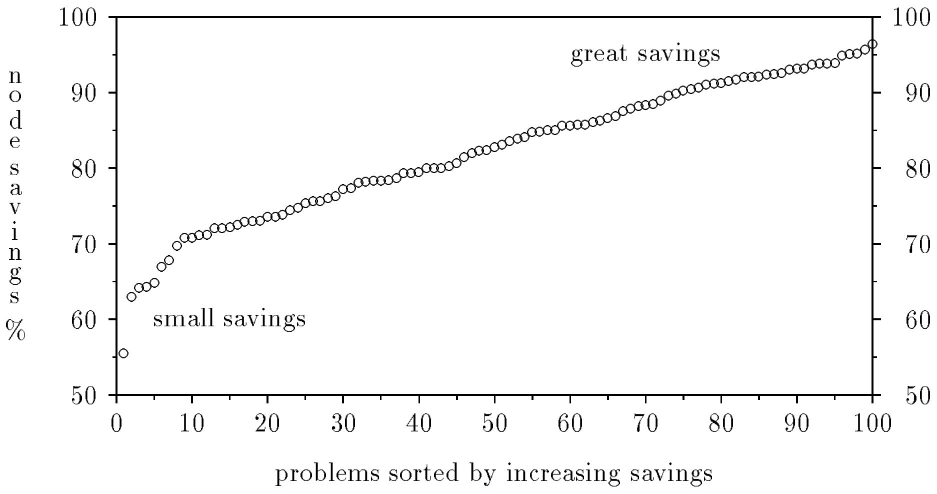

Figure 4: Node savings of MPD heuristic (all-solutions-case)

Figure 4 shows the reduced node expansions for the all-solutions-case, sorted by increased node savings to the original MD heuristic. Savings vary between 55.45 % (problem 23) and 96.42 % (problem 57). Counting the savings on the average this results in 82.12 %. The single savings vary much, yielding the mean deviation to 8.99 . Looking at the sum of the total number of node generations we get an even better figure of 88.12 % node savings.

Fig. 5 shows that the savings do not increase with increasing problem size. There seems to be no relationship allowing conclusions with respect to the problem size.

Turning now to the *first-solution-case*, we observe similar results. The improvement differs from 56.29 % (problem 23) to 96.52 % (problem 57), leading to an average value of 82.05 % with a mean deviation of 9.02. We omit the pictures here because they are very similar to the all-solution-case.

We should now have a look more precisely to the effects of the PD in the IDA* algorithm. At the very beginning we might save one, two

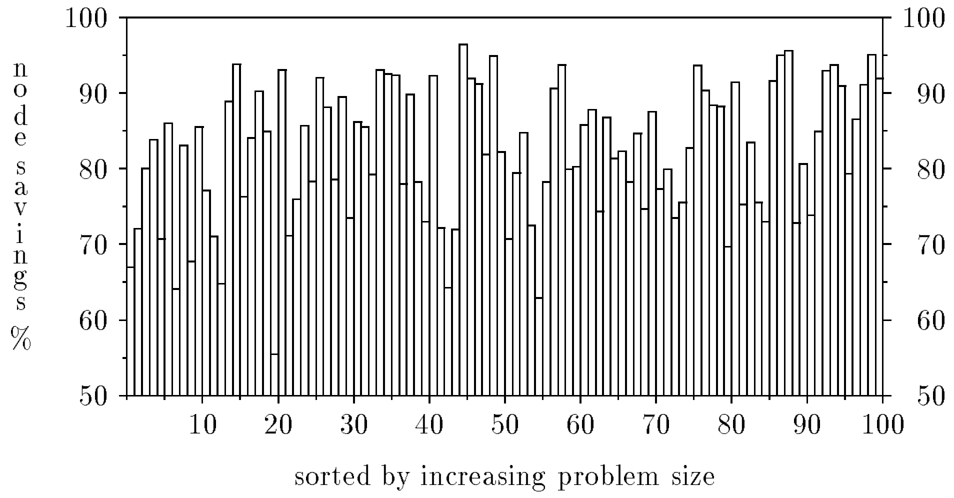

Figure 5: Node saving and problem size (all-solutions-case)

or more iterations, depending on the pairs in the start configuration. If a move puts a tile in a new pair relationship, MPD increases by 2. Consequently, a node cutoff may occur earlier than with pure MD, so that now moves that even decrease the heuristic estimation need not be generated. Mainly due to this fact, MPD saves more than 80 % as compared with MD.

To show the significance of pairs we determined the part of search nodes (in %) with at least one pair for every problem. Arranging these values in increasing order Fig. 6 shows the savings for the problems. As expected, the best savings result from problems that cause the generation of many nodes with pairs during the search.

Interestingly, on the average 42.45 % of the nodes in the 100 puzzle instances have at least one pair with a minimum of 14.15 % and a maximum of 86.14 %. Only about 8 % of the random configurations have more than one pair. Two or more pairs appear very rarely ($\leq$ 1 %). Every 11th node generation produces a new pair relation removing other ones accordingly. 29 of the test instances have no pairs at the beginning, 47 have 1 pair, 20 with 2 pairs and 4 got 3

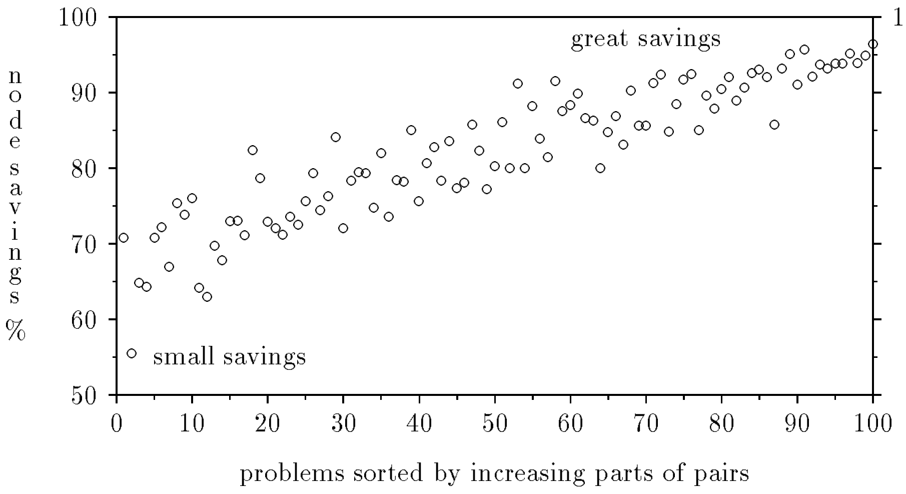

Figure 6: Pairs and Savings

pairs.

Note that we did not exploit the complete power of the PD in our implementation because of the small size of the 4×4-tray, where the probability for pairs is low. Additional overhead for checking *all possible* pairs (e.g. tile x in a row and column pair relationship at the same time) did not pay off in our implementation. In total, we achieved a 50 % CPU time reduction with the MPD heuristic.

5 Conclusions and Future Research

With our Manhattan Pair Distance heuristic we improved the best-known heuristic function for the 15-puzzle, giving a 80 % reduction in the node count. These savings reduce the CPU time of the fastest sequential program by one half. Greater puzzle sizes, e.g. 4×5 or 5×5, offer still better prospects for our MPD, because there is a higher probability of pair occurrences. Nevertheless we see no complete solution of greater puzzle problems using only the MPD.

Our future work therefore includes the parallelization of IDA* with MPD an a massively parallel MIMD machine.

Also, we will follow the idea of joining heuristic functions to a new composed heuristic function, yielding better results than the single original ones. This will be done for other problem domains to achieve new and improved results.

6 Acknowledgments

Many thanks to Hildegard Hügemann for the joint work providing basic results for this paper and to Tony Marsland for discussions. Our special thanks to Alexander Reinefeld for motivation to work out the ideas of this paper, his proposals and comments.

References

[1] Bernard Bauer and Hildegard Krieter. *Parallelisierung probleml"osender Methoden am Beispiel des $n \times n$ Puzzles*, master thesis, University Paderborn, July 1988.

[2] Wm. Woolsey Johnson. *Notes on the 15-puzzle (1).* American Journal Mathematics (1879), 397-399.

[3] Richard E. Korf. *Depth-first iterative-deepening: An optimal admissible tree search.* Artificial Intelligence 27(1985), 97-109.

[4] Richard E. Korf. *Iterative Deepening A^* an optimal admissible tree search.* Proc. 9th Int. Joint Conf. on AI, Los Angeles, CA(1985), 1034-1036.

[5] N.J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing, Palo Alto, CA, (1980).

[6] J. Pearl. *Heuristics. Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, (1984)

[7] D. Ratner and W. Warmuth. *Finding a shortest solution for the $N \times N$ extension of the 15-puzzle is intractable.* AAAI-86, 168-172.

[8] A. Reinefeld, *Complete Solution of the Eight-Puzzle and the benefit of node-ordering in IDA**, Procs. Int. Joint Conf. on AI, Chambery(1993), Savoi, France,248-253.

[9] P.D.A. Schofield. *Complete Solution of the Eight-Puzzle.* Machine Intelligence(1965), 1:125:133.

[10] Edward L. Spitznagel. *A new look at the fifteen puzzle.* Math. Magazine(40,Sept.-Oct. 1967), 171-174