

FINDING A SHORTEST SOLUTION FOR THE $N \times N$ EXTENSION OF THE 15-PUZZLE IS INTRACTABLE

Daniel Ratner and Manfred Warmuth

Computer & Information Sciences
University of California Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

The 8-puzzle and the 15-puzzle have been used for many years as a domain for testing heuristic search techniques. From experience it is known that these puzzles are "difficult" and therefore useful for testing search techniques. In this paper we give strong evidence that these puzzles are indeed good test problems. We extend the 8-puzzle and the 15-puzzle to a $n \times n$ board and show that finding a shortest solution for the extended puzzle is NP-hard and thus computationally infeasible.

We also present an approximation algorithm for transforming boards that is guaranteed to use no more than $c L(SP)$ moves, where $L(SP)$ is the length of the shortest solution and c is a constant which is independent of the given boards and their size n .

I. INTRODUCTION

For over two decades the 8-puzzle and the 15-puzzle have been a laboratory for testing search methods. Michie and Doran used these games in their general problem-solving program, called Graph Traverser [DM66]. Pohl used the 15-puzzle in his research on bi-directional search and dynamic weighting [Po77]. Recently Korf used these puzzles as examples for the Macro-Operators [K85a] and for IDA* [K85b]. Judea Pearl used the 8-puzzle throughout the first half of his Heuristics book as one of the main examples [Pe84]. Also, these puzzles were used for testing the performance of some learning algorithms [Re83].

The main reasons for selecting these problems as workbench models for measuring the performance of searching methods are:

- 1) There is no known algorithm that finds a shortest solution for these problems efficiently.
- 2) The problems are simple and easy to manipulate.
- 3) The problems are good representatives for a class of problems with the goal of finding a relative short path between two given vertices in an undirected graph.
- 4) The size of the search graph is exponential in n even though the input configurations can be described easily ($O(n^2)$).
- 5) The search graph can be specified by a few simple rules.

Certainly, if there existed simple efficient algorithms for finding a shortest solution for these problems, then heuristic approaches would become superfluous. Thus we need to give a convincing argument that no such algorithm exists. This is accomplished by using complexity theory. We show that finding the shortest solution for a natural extension of the 8-puzzle and the 15-puzzle is NP-hard. Thus unless $P=NP$,

which is considered to be extremely unlikely, there is no polynomial algorithms for finding a shortest solution. Of course, since the number of distinct configurations in the 8-puzzle and the 15-puzzle are finite, theoretically (and practically for the 8-puzzle) one can find shortest solutions for all the possible inputs by analyzing the whole search graph. To get problems of unbounded size we extend the problem to the $n \times n$ board ((n^2-1) -puzzle).

The aim of the (n^2-1) -puzzle is to find a sequence of moves which will transfer a given initial configuration of an $n \times n$ board to a final (standard) configuration. A *move* consists of sliding a *tile* onto the empty square (*blank tile*) from an orthogonally adjacent square.

We will show that the following decision problem (*nPUZ*) is NP-complete:

Instance: two $n \times n$ boards and a bound k .

Question: is there a solution for transforming the first board into the second board requiring less than k moves?

The pebble games of [KMS84] can be viewed as a direct generalization of the *nPUZ* problem. Rather than moving tiles in the planar grid, they allow general graphs with an arbitrary number of empty spaces. They address the question of reachability, i.e. whether a final configuration is reachable from an initial configuration by moving pebbles to adjacent empty spaces. It was shown that the general reachability problem can be decided in polynomial time.

The *nPUZ* problem is case where reachability is easy. We address the complexity of reaching the final configuration from the initial configuration in a small number of moves.

In the *nPUZ* problem we relocate tiles. The relocation task, even without the specific rigid rules of the game, is the essence of the intractability. In the *nPUZ* problem we have additional restrictions that makes its proof of NP-completeness very difficult. Therefore we first show the intractability of a relocation problem. This problem, the *REL* problem, captures the hardness of *nPUZ* and is less restrictive and easier to prove NP-complete. The *REL* problem is specified as follows:

Instance: A planar directed graph $G(V, E)$ where each $e \in E$ has capacity 0 or 2, a set X of elements, and an initial and final configuration. A configuration specifies the location of each element of X at the vertices of V .

Question: Is there a relocation procedure that ships the elements of X from their initial configuration to their final configuration such that the procedure moves along each $e \in E$ exactly once and along each edge it never ships more elements than allowed by its capacity?

The *nPUZ* and *REL* problems can be viewed as robotics problems: A robot needs to efficiently relocate objects in the plane.

The NP-completeness proof of *nPUZ* will simulate the simpler proof for *REL*. The graph is mapped onto the board of the puzzle problem. The vertices and edges will correspond to certain areas of board. The elements and the capacities are encoded by the arrangements of tiles in these areas in the start and final configuration.

Since finding the shortest solution is NP-hard we would like to know how close the shortest solution can be approximated. We show that finding a solution that is an additive constant away from the optimum is also NP-hard. However we have positive results for approximating the optimal solution by a multiplicative constant. We only give a proof for a high multiplicative constant. However we suspect that the algorithm outputs a solution that is not more than twice the optimal. It is an open problem to find the algorithm with the best possible constant. Note that this type of algorithm finds reasonable solutions even if n is large (around 100). The research based on search methods only addresses the cases of $n \leq 4$. The best solutions will be produced by a combined approach, as suggested in [RP86]: use an approximation algorithm and then do local optimization of the approximate solution employing search methods.

We suggest to apply our approach to other puzzles, like the Rubik's cube, which are studied extensively in the AI-literature. Is the problem of finding a shortest solution for the n -dimensional Rubik's cube NP-hard? Are there polynomial time approximation algorithms for this puzzle that approximate the optimal solution by a multiplicative constant?

II. THE NP-COMPLETENESS OF *REL*

In this section we prove that *REL* problem is NP-complete, i.e. relocating elements that reside in vertices of a planar graph via an Eulerian path is NP-complete. We prove that *REL* and *nPUZ* are NP-complete by reducing a special very symmetric version of the satisfiability problem to *nPUZ*. This version is called 2/2/4-SAT and is defined as follows: each clause contains four literals; each variable appears four times in the formula, twice negated and twice not negated; the question is whether there is a truth setting for the formula such that in each clause there are exactly two true literals. In the complete paper we give a standard NP-completeness reduction for 2/2/4-SAT.

Theorem 1: *REL* is NP-complete.

Proof: Let $U = \{u_1, u_2, \dots, u_m\}$ be a set of variables and $C = \{c_1, c_2, \dots, c_m\}$ be a set of clauses defining an arbitrary instance of 2/2/4-SAT. From this instance we will construct an instance of *REL*. An instance of *REL* is a graph $G(V, E)$ with capacities (0 or 2) for each $e \in E$, a set X of elements, an initial configuration (called B_1), and a final configuration (called B_2). First we start with the description of the graph and later we define the configurations.

The graph (Figure 2) consists of $5m+2$ vertices and $12m-3$ edges. The vertices are divided to 4 groups. The first group is built up from m diamonds of 4 vertices each. The i -th diamond which is shown in Figure 1 corresponds to the

variable u_i . This diamond contains the vertices: top_i , nu_i , bot_i , and \overline{nu}_i .

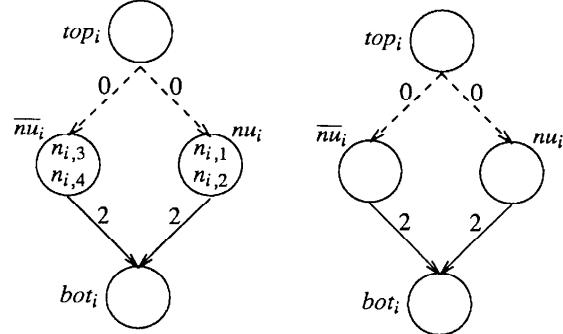


Figure 1: The i -th diamond in B_1 and B_2 .

The second group is the single vertex TC (stands for truth collection). The third group is the single vertex FC (stands for false collection). The fourth group consists of m vertices. The i -th vertex of this group, called nc_i , corresponds to the i -th clause in the boolean formula of the 2/2/4-SAT instance.

The directed edges connecting the vertices and the capacities of the edges are specified in Figure 2. Note that there is a special edge of capacity zero from nc_m to top_1 .

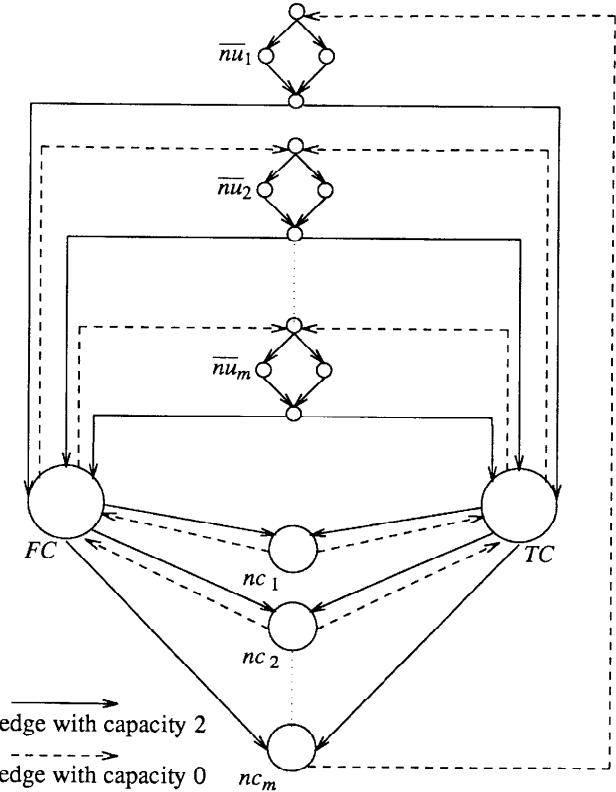


Figure 2: The graph of the *REL* instance.

To complete the definition of the instance of *REL* we need to specify the elements and their initial and final locations in the graph. The set of elements X consists of $4m$ elements. Recall that in 2/2/4-SAT each variable occurs

twice negated and twice unnegated. There is an element for each of the 4 occurrences: $n_{i,1}$ and $n_{i,2}$ correspond to the two appearances of u_i in C , and $n_{i,3}$ and $n_{i,4}$ correspond to the two appearances of \bar{u}_i in C .

In B_1 the elements are located in the diamonds as specified in Figure 1. All the remaining vertices contain no elements. In B_2 all elements are in the vertices that correspond to the clauses. The 4 elements that are associated with the 4 literals of the i -th clause appear in vertex nc_i . This completes the definition of the instance of REL . The following two claims complete the proof of the theorem.

Claim 1: If there is a truth assignment $f : U \rightarrow \{T, F\}$ that satisfies the 2/2/4-SAT instance then there is a relocation procedure along an Eulerian path that shifts B_1 to B_2 .

Proof: The proof is constructive. First we ship all the $n_{i,j}$ elements that correspond to true literals from their vertices in B_1 to TC vertex. This collection is done by the following loop:

```

for i := 1 to m do begin
    if f(ui) = T
        then begin
            move along (topi,nui);
            move along (nui,boti) with ni,1 and ni,2;
            move along (boti,TC) with ni,1 and ni,2;
        end
        else begin
            move along (topi,nūi);
            move along (nūi,boti) with ni,3 and ni,4;
            move along (boti,TC) with ni,3 and ni,4;
        end { if };
        if i ≠ m then move along (TC,topi+1);
    end.

```

When the above loop is finished, then the vertex TC contains $2m$ elements. Each diamond contributes exactly two elements. The two elements from the i -th diamond are either $n_{i,1}$ and $n_{i,2}$ (from nu_i) or $n_{i,3}$ and $n_{i,4}$ (from $nū_i$).

The next step drops the $2m n_{i,j}$ elements that are in TC into the nc_i vertices they belonged to in B_2 . As mentioned above, these $2m n_{i,j}$ elements correspond to the $2m$ true literals. Since there is a truth assignment for the 2/2/4-SAT instance, it follows that two $n_{i,j}$ elements, that appear in each clause vertex nc_i in B_2 are now in TC . These elements are dropped into their clause vertices by the following loop:

```

for i:=1 to m do begin
    move along (TC,nci) with the two ni,j elements
        that are in nci in B2;
    if i ≠ m then move along (nci,TC);
end.

```

As a result of the above segment, each nc_i vertex receives the two $n_{i,j}$ elements that correspond to the true literals.

Now we move along (nc_m, top_1) . From this point we repeat the two loops given above. In the first loop we collect all the $n_{i,j}$ elements that correspond to false literals into the FC vertex. This is done by traversing all the edges of the diamonds that have not been traversed in the first pass and by traversing all the edges that connect FC with the diamonds. Once the first loop in this second pass is completed, the $2m n_{i,j}$ elements that correspond to the $2m$ false literals are in FC . In the second loop of the second pass, the algorithm

drops the $2m n_{i,j}$ elements from FC into their appropriate nc_i vertices in B_2 . Once the second pass is completed the arrangement of the $n_{i,j}$ elements in the graph is as prescribed in B_2 . Observe that each edge is traversed exactly once and the number of elements moved through each edge always equals the capacity of the edge.

Claim 2: If there is a relocation procedure that ships the elements from B_1 to B_2 along an Eulerian path then there is a truth assignment $f : U \rightarrow \{T, F\}$ that satisfies the 2/2/4-SAT instance.

Proof: We need to ship the four $n_{i,j}$ elements from their initial locations in the i -th diamond to the clause vertices (the nc_k) they belong to in B_2 . The $n_{i,j}$ elements must pass through bot_i . There are only two edges (bot_i, TC) and (bot_i, FC) outgoing from bot_i . Both edges have capacity 2. This means that when the procedure moves along (bot_i, TC) and (bot_i, FC) it must carry 2 elements each time. Furthermore, the first time the procedure ships two elements to bot_i they must be either the pair $(n_{i,1}, n_{i,2})$ or the pair $(n_{i,3}, n_{i,4})$. Now the procedure must continue to move along with these two elements. Thus, for each i , $1 \leq i \leq m$, the procedure that relocates the elements ships the pair $(n_{i,1}, n_{i,2})$ along (bot_i, TC) or along (bot_i, FC) .

Let us define the truth assignment $f : U \rightarrow \{T, F\}$ as follows:

```

f(ui) = T if the procedure ships the pair (ni,1,ni,2)
    along (boti,TC).
f(ui) = F if the procedure ships the pair (ni,1,ni,2)
    along (boti,FC).

```

Note that if $f(u_i) = T$ (respectively F) then the procedure ships the pair $(n_{i,3}, n_{i,4})$ along (bot_i, FC) (respectively (bot_i, TC)).

We proceed to show that the above truth assignment satisfies the requirements of the 2/2/4-SAT instance. There are two ingoing edges to each nc_i vertex, each of capacity two. There is no way to ship elements from TC to FC or vice versa (see Figure 2). Thus the procedures ships from TC exactly two elements to each of the nc_i vertices. According to the definition of f these elements correspond to true literals. The other two elements that arrive at each nc_i vertices are from FC , which means that they correspond to false literals. This completes the proof of Claim 2 and Theorem 1. \square

III. THE NP-COMPLETENESS OF $nPUZ$

In this section we will sketch a reduction of the 2/2/4-SAT problem to the $nPUZ$ problem. Given an instance of 2/2/4-SAT we define a corresponding instance of $nPUZ$. This instance (and the whole reduction) is similar to the instance of REL used in the previous section. We will map the graph of Figure 2 onto the board. The instance of $nPUZ$ consists of two $n \times n$ board configurations B_1 (the initial configuration), B_2 (the final configuration) and an integer k which is an upper bound on the number of moves that can be used to transform B_1 to B_2 . To simulate the graph of Figure 2 we have to capture the notions of vertices, edges, elements, relocation, moving along an edge and capacity of an edge. Each vertex in the graph of Figure 2 corresponds to a square of locations. Edges are identified as stripes (horizontal, vertical, or a pair of both) of locations that connect the

vertices. Each element of X corresponds to a specific tile on the board. The tiles which correspond to the elements appear in different locations on board B_1 than on board B_2 . As in the instance of *REL* the element tiles are in the diamonds on B_1 and in the squares of the clauses in B_2 . Moving these tiles to their destination in B_2 corresponds to relocating the elements in the graph of the *REL* problem. Until now, the analog between the components in the *REL* problem and the corresponding components in the game are straight forward. An outline of how the graph is mapped onto the board is given in Figure 3. The main difference is a 45 degree counterclockwise rotation. Note that the lines of Figure 3 represent "thin" stripes of locations. The arrangements of the tiles outside the squares of the vertices and outside the stripes of the edges are the same on B_1 and B_2 . Note that all the names of the tiles on the board are distinct. Thus the configurations B_1 and B_2 are equivalent w.r.t. renaming of tiles and only the relative location of equally named tiles on B_1 and B_2 is important.

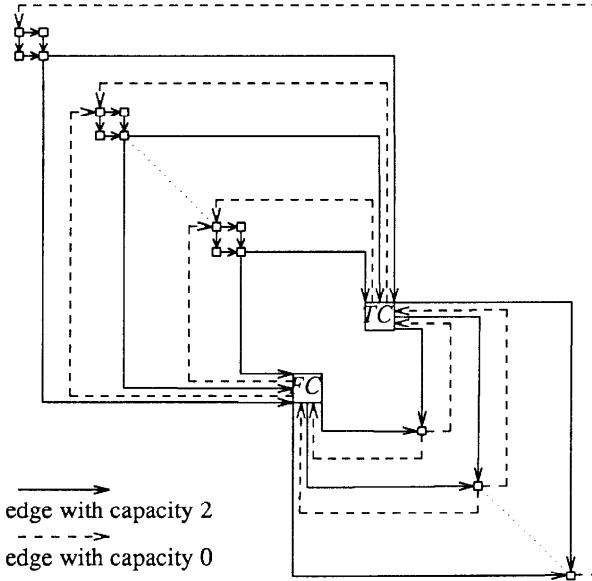


Figure 3: The locations in which B_1 and B_2 differ.

We still need to show the analog to "move along an edge". In the game, the tiles can be shifted to any location, and they are not tied to specific squares and stripes of locations. How can we force the tiles that correspond to the elements of the X to move only along the stripes of the edges? How can we force these tiles to move along a stripe exactly once? To realize the notion of capacity we need to guarantee that exactly two element tiles (in addition to the blank tile) move along the stripes of capacity two, and zero element tiles, i.e. only the blank tile, move along the stripes of capacity zero. To overcome the above difficulties we carefully arrange the tiles within the edges. The vertices and edges are the only locations in which B_1 and B_2 differ.

Edges either have capacity zero or capacity two. They are stripes following the outline of Figure 3. The edges of capacity zero are stripes of width 1 and the edges of capacity two are stripes of width 3. The tiles within the edges are arranged differently in B_1 and B_2 . Recall that each edge has a

direction. For the edges of capacity zero the tiles of B_2 are shifted one location backward relative to their location in B_1 . This will guarantee that the blank tile has to move through this edge to achieve the rearrangement of the edge. The overall bound on the number of moves will assure that this can happen only once.

The rearrangement of the edges of capacity two is given in Figure 4, 5 and 6. The figures show how to move two tiles (x and y) together with the blank tile through a stripe edge of width 3 and length l (the edge is the portion between the double bars). Figure 4 shows the arrangement of the tiles on the edge in B_1 and Figure 6 the same for B_2 .

α_{-2}	α_1	α_0	α_1	α_2	α_3	...	α_l	α_{l+1}	α_{l+2}	α_{l+3}
x	y		β_1	β_2	β_3	β_l	β_{l+1}	β_{l+2}	β_{l+3}
γ_{-2}	γ_1	γ_0	γ_1	γ_2	γ_3	γ_l	γ_{l+1}	γ_{l+2}	γ_{l+3}

Figure 4: The arrangement of an edge with capacity 2 in B_1 .

Figure 5 (below) is produced from Figure 4 by advancing x and y one location to the right. This is accomplished by moving the blank tile: left, left, up, right, right, right, down, left, left, down, right, right, right, up.

α_{-1}	α_0	α_1	β_1	α_2	α_3	...	α_l	α_{l+1}	α_{l+2}	α_{l+3}
α_{-2}	γ_1	x	y		β_3	β_l	β_{l+1}	β_{l+2}	β_{l+3}
γ_{-2}	γ_0	γ_1	γ_2	β_2	γ_3	γ_l	γ_{l+1}	γ_{l+2}	γ_{l+3}

Figure 5: The arrangement after advancing x and y once.

If we apply the above sequence $\frac{l-1}{2} + 3$ times then we end up with the following arrangement.

α_{-1}	α_0	β_1	α_2	β_3	...	β_l	α_{l+1}	α_{l+2}	α_{l+3}
α_{-2}	γ_1	α_1	γ_2	α_3	α_l	γ_{l+1}	x	y
γ_{-2}	γ_0	γ_1	β_2	γ_3	γ_l	β_{l+1}	γ_{l+2}	β_{l+3}

Figure 6: The arrangement of an edge with capacity 2 in B_2 .

For all edges of capacity 2 the tiles on B_2 are rearranged as if two pieces and the blank tile moved through the edge on B_1 in the prescribed fashion. Note that the procedure specified in the three figures uses the minimal number of moves, i.e. there is no procedure that moves x and y through the edges using a smaller number of moves. Also the procedure uses no more moves than the manhattan distance between the arrangement of e on B_1 and B_2 . This rearrangement can not be accomplished efficiently by any other shifting procedure, i.e. by moving twice from the beginning to the end of the edge or by moving through the edge with less or more than 2 non-blank tiles. The number of moves required by any other shifting procedure exceeds the manhattan distance and the number of additional moves required is proportional to the length of the edge. Since the bound on the overall number of moves will be tight, B_1 must be rearranged efficiently to achieve B_2 , i.e. each edge of capacity 2 is traversed exactly once while shipping exactly 2 non-blank tiles along the edge.

A detailed proof is given in the complete paper. We haven't specified the rearrangements of the vertices and the corners of the edges. We define k to be equal to the number

of moves required by traversing each edge exactly once with the specified number of non-blank tiles plus enough freedom to achieve the rearrangements in the corners of the edges and in the vertices. This freedom is much smaller than the number of moves required to travel along an edge with only the blank tile.

It might seem that B_2 uniquely determines the rearrangement procedure. However note that the rearrangement of the edges (Figure 4 and 6) is independent of the element tiles moved through the edge. The element tiles are located in the same vertices as in the instance of *REL*. They are to be moved from the diamonds to the clauses. Half of the element tiles are gathered in *TC* and these tiles correspond to the true literals. The other half is gathered in *FC*. The reduction is identical to the reduction of *REL*.

IV. AN APPROXIMATION ALGORITHM

Since finding a shortest solution is NP-hard we would like to know how close a shortest solution for the (n^2-1) -puzzle can be approximated. We can prove that finding a solution that is an additive constant larger than the optimum is also NP-hard. We simply use the reduction of the previous section except that we enlarge the length of each edge by four times the additive constant.

In this section we sketch the main ideas of a polynomial approximation algorithm that approximates the optimal solution by a multiplicative constant. We chose to present a simplified version of our algorithm. Therefore the multiplicative constant is not as low as possible. The main point is that such an approximation algorithm exists. Also for simplicity we assume that the blank tile resides in the same location (n,n) in both configurations.

Let us denote B_1 as a permutation of B_2 and then decompose the permutation into disjoint cyclic permutations. The decomposition can be produced in time which is linear in the size of the permutation.

Assume that there is only one cycle, $(l_0, l_1, \dots, l_{c-1})$, denoting the fact that the tile located at l_i in B_1 is located at $l_{(i+1) \bmod c}$ in B_2 . There are two simple lower bounds on the length of the optimal solution: $d((n,n), l_0)$ and $\sum_{i=0}^{c-2} d(l_i, l_{i+1})$, where $d(l, l')$ is the manhattan distance between the locations l and l' . The following procedure requires at most $2d((n,n), l_0) + 20 \cdot \sum_{i=0}^{c-2} d(l_i, l_{i+1})$ moves, which is at most 22 times the length of the optimal solution: the blank tile moves from (n,n) to l_0 ; the pieces at locations l_0, l_1, \dots, l_{c-2} are shifted one at a time to the locations l_1, l_2, \dots, l_{c-1} , respectively; this shifting process has the side effect that a large number of tiles on the path between the locations are shifted one or two places from their origin (see figures 4, 5 and 6); now the tile at l_{c-1} is moved along the path $l_{c-1}, l_{c-2}, \dots, l_0$ to its destination at l_0 ; while relocating the tile of l_{c-1} the side effects of the shifting process are undone; finally, the blank tile moves from l_0 to (n,n) .

In the case where there is more than one cycle, each cycle C contributes its sum of distances around the cycle

(denoted by $S(C)$) to the lower bound. Thus the second lower bound becomes $\sum_{all\ cycles\ C} S(C)$. The first lower bound

$d((n,n), l_0)$ is replaced by the cost of the minimum spanning tree given below. We can view the l_0 's of each cycle and (n,n) as nodes in a complete graph, where the cost of each edge is the manhattan distance between the corresponding locations. Clearly the cost of the minimum spanning tree is a lower bound for the length of the optimal solution. (Note that the minimum spanning tree can be constructed in $O(n^4)$ time.) In the case where there is only one cycle, the vertices (n,n) and l_0 , and the path between them represent the minimum spanning tree. If we have more than one cycle we connect the l_0 's of all the cycles according to the edges of the minimum spanning tree. The procedure for the general case: the blank tile moves along the locations that correspond to edges of the spanning tree as if it traverses the tree; whenever it reaches an l_0 location the shifting process in a cycle is executed; backtracking from a child to a parent in the traversal corresponds to undoing the changes made on the edge that connects the child with its parent.

There are two difficulties in the above description. The first one is that if the number of locations in a cycle is even we can't completely fix the cycle because of the group properties of *nPUZ*. The second difficulty is that when we shift tiles along a path we might increase the manhattan distance between the tiles of some cycle that crosses the path, and then the second lower bound is improper. In the complete paper we show how we overcome these difficulties without additional moves.

REFERENCES

- [DM66] Doran, J. and Michie, D., "T Experiments with the graph traverser program" *Proc. of the Royal Society (A)*, No. 294, pp.235-259, 1966.
- [K85a] Korf, R. E., *Learning to solve problems by searching for Macro-Operators*. *Research Notes in Artificial Intelligence 5*, Pitman Advanced Publishing Program, 1985.
- [K85b] Korf, R. E., "Iterative-Deepening-A^{*}: An Optimal Admissible Tree Search," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Vol. 2, pp. 1034-1035, 1985.
- [KMS84] Kornhauser, D., Miller, G. and Spirakis, P., "Coordinating Pebble Motion on Graphs, The Diameter of Permutation Groups, and Applications," *25th FOCS*, pp. 241-250, 1984.
- [Pe84] Pearl, J., *Heuristics. Intelligent search strategies for computer problem solving*, Addison-Wesley Publishing Company, 1984.
- [Po77] Pohl, I., "Practical and theoretical considerations in heuristic search algorithms," in *Bernard Meltzer and Donald Michie (editors) Machine Intelligence 8*, pp. 55-72, American Elsevier, New York, 1977.
- [Re83] Rendell, L. A., "A new basis for state-space learning systems and a successful implementation," *Artificial Intelligence*, Vol. 20, pp. 369-392, 1983.
- [RP86] Ratner, D. and Pohl, I., "Joint and LPA^{*}: Combination of Approximation and Search," *to appear in the Proceedings of AAAI-86*, 1986.