On Sliding Block Puzzles

Filip R. W. Karlemo
Tellabs Oy
Porarinkatu 1
02600 Espoo, Finland

Patric R. J. Östergård*
Department of Computer Science and Engineering,
Helsinki University of Technology,
P.O. Box 1100, 02015 HUT, Finland

Abstract

A graph of a puzzle is obtained by associating each possible position with a vertex and by inserting edges between vertices iff the corresponding positions can be obtained from each other in one move. Computational methods for finding the vertices at maximum distance δ from a vertex associated with a goal position are presented. Solutions are given for small sliding block puzzles, and methods for obtaining upper and lower bounds on δ for large puzzles are considered. Old results are surveyed, and a new upper bound for the 24-puzzle is obtained: $\delta \leq 210$.

1. Introduction

In the early 1980s, it was impossible to avoid hearing about Rubik's cube, a puzzle that became very popular all over the world. Very soon, mathematicians became interested in this puzzle, and several books have been written on the subject (for example, [4]). Another popular—and much older—puzzle is the 15-puzzle, which was invented by Sam Loyd in the 19th century. This puzzle, and its variants, will be considered in this paper. The 15-puzzle is discussed in many books on games [1], discrete mathematics [6], and computer algorithms [8, 12]. In particular, it has turned out to be

*The research was supported by the Academy of Finland and the Alfred Kordelin Foundation.

a good test case for search methods in artificial intelligence (path planning and scheduling problems); see [5] and its references.

A puzzle is solved by restoring a given position to a goal position. A mathematician might not be satisfied with just a solution, but might want to determine the minimum number of moves required to solve the puzzle. Furthermore, it is very natural to try to find the "most difficult" initial positions, that is, the positions that require most moves to be restored to the goal position; in this paper, this problem is considered.

The paper proceeds as follows. In Section 2 the 15-puzzle and its variants are discussed and mathematical notations are introduced. Positions are mapped to vertices of a graph, which makes it possible to discuss the problem in graph theoretical terms. Computational methods for analyzing puzzles are considered in Section 3. It turns out that it is the memory size of the computer used rather than time available that sets the limits of how large puzzles can be analyzed. Solutions of sliding block puzzles of size up to 3×4 are given in Section 4. Methods for obtaining bounds on the number of moves required to solve the 15-puzzle and larger puzzles are discussed in Section 5.

2. The 15-Puzzle and Its Variants

The 15-puzzle is a 4×4 puzzle with 15 blocks (numbered 1 to 15) and one empty square. A move consists of sliding a block adjacent to the empty square into that place. The aim is to restore an initial position to the following goal position (other different but equivalent numberings of the blocks can often be seen):

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

Methods have been developed for finding solutions of the 15-puzzle with as few moves as possible from a given initial position; see, for example, [3, 5, 7, 10, 11, 13, 16]. This is a difficult problem; in fact, the $n \times n$ extension of it is NP-hard [15].

In the general case, we consider a $p \times q$ puzzle. Let $n = pq$. The squares of such a puzzle are here numbered row-wise from 1 to n, with 1 in the upper left corner and n in the lower right corner. The goal position is achieved when block i is in square i for $1 \leq i \leq n-1$ and square n is empty. (It is important to distinguish between the numbering of the blocks and the numbering of the squares.)

There are $n!$ possible positions of a $p \times q$ puzzle. Now a graph G with $n!$ vertices can be constructed by associating each possible position with a vertex, and by inserting edges between two vertices iff the corresponding positions can be obtained from each other in one move (since a move can always be reversed, we let G be undirected).

Properties of a puzzle can now be discussed in graph-theoretic terms. For example, an initial position can be restored to the goal position exactly when its vertex in G is in the same connected component as the vertex of the goal position (the *goal vertex*). This matter is thoroughly discussed in [20].

For a $p \times q$ sliding block puzzle, it was shown more than 100 years ago [9, 19] that not all $n!$ possible positions can be restored to the goal position. Namely, when the empty square is in a given place, to obtain another position with the empty square in the same place clearly requires an even number of moves. If the empty square is identified as block number n, a move is a transposition of the blocks, so only $n!/2$—which is the order of the alternating group A_n—positions can be restored to the goal position.

3. Computing Maximum Distances in a Puzzle Graph

We shall now discuss computational methods for finding the vertices in the same connected component of G that are at maximum distance δ from the goal vertex. This is done by an exhaustive search. Since there are $n!/2$ vertices in the connected component, it is clear that such a method works only for small values of n.

The search method used is a breadth-first search. That is, we start with the goal vertex, which is at distance 0 from itself. The set of vertices at distance $d+1$ is calculated by considering the vertices at distance d from the goal vertex and checking which of its (at most 4) adjacent vertices have not been encountered earlier in the search. The adjacent vertices are actually at distances either $d-1$ or $d+1$ from the goal vertex. This procedure is repeated until all $n!/2$ vertices of the component have been encountered. (For $n \times n$ puzzles, the number of vertices that have to be searched can be reduced by another factor of approximately 2 due to symmetry of the goal position.)

In the search, we are primarily interested in the distances from vertices to the goal vertex. Thus, to save memory, we do not save the edges between vertices. We use three data structures: one table with vertices (positions) at distance d, one table with vertices (positions) at distance $d+1$, and one table that for each vertex indicates whether it has been encountered earlier. The two former tables can be saved in a secondary memory, since, for each

level, they are written and read once and sequentially. The last table, the *main table*, should, however, be in the primary memory, since its elements are accessed in a nonsequential order.

For the 3×4 puzzle, the largest puzzle that we have solved completely, the number of entries in the main table is 239500800. (Actually, to obtain Theorem 1 to be presented later, larger puzzles than the 3×4 puzzle with different shapes have been solved.) We use one bit of memory for each entry, that is, the total memory requirement is approximately 30 MB—which many modern computers can manage. The 15-puzzle, however, requires about 1300 GB, which certainly still is beyond reach.

Let C_i be the number of the block in square i (the empty square is block n). For the main table we need a bijective function from $(C_1\ C_2 \cdots C_n)$ to an index $0 \le I \le n!/2-1$. We modify a function that maps permutations of the elements $\{1, 2, \ldots, n\}$ bijectively to the set of integers $\{0, 1, \ldots, n!-1\}$; for such functions, see [17] or any other book on combinatorial algorithms. Many of these functions have the property that for any i, the first $n-i$ elements of a permutation are mapped to an integer $0 \le a' \le n!/i!-1$, the last i elements are mapped to an integer $0 \le b' \le i!-1$, and $I = a' \cdot i! + b'$. We now choose $i = 3$, $a = a'$, and let $I = 3a + b$, where b indicates the place of the largest of the last three elements (which possibly is block n, the empty square). The order of the other two of the last three elements is then fixed.

It is clear that, as the total number of moves tried in a run (and thus index calculations) is at most $3n!/2$ (we save the move to a new position, so that we need not try its inverse) and the total number of memory bits required for the main table is $n!/2$, it is the memory size that sets the limits of how large puzzles can be analyzed with this approach.

4. Complete Solutions of Small Puzzles

This section is mostly a survey of old results. Although we do not have references to all those results, they have all been presented and discussed in several forums (for example, in electronic newsgroups). The fact that these results, which are based on computer searches, have been presented by several independent researchers, indicates that they are correct.

The smallest possible nontrivial sliding block puzzle is the 2×2 puzzle. Since it has $4!/2 = 12$ possible positions, the case is easily solved by hand. We can also argue as follows. The empty cell is always in a corner, so there are always two possible moves. Since the graph of the puzzle is connected and is regular of degree 2, it is a circuit graph. Thus, $\delta = 12/2 = 6$.

For larger puzzles, the use of computer is inevitable. We shall here give a complete solution of puzzles of sizes 2×3, 2×4, 3×3, and 3×4. The

goal positions of these puzzles are as follows:

1	2	3
4	5	

1	2	3	4
5	6	7	

1	2	3
4	5	6
7	8	

1	2	3	4
5	6	7	8
9	10	11	

The results of the calculations can be found in Table I, where the number of vertices at each depth is given. The values of δ for the puzzles considered are obtained directly from the table.

TABLE I. Solutions of small puzzles

Depth	2×2	2×3	3×3	2×4	3×4
0	1	1	1	1	1
1	2	2	2	2	2
2	2	3	4	3	4
3	2	5	8	6	9
4	2	6	16	10	20
5	2	7	20	14	37
6	1	10	39	19	63
7		12	62	28	122
8		12	116	42	232
9		16	152	61	431
10		23	286	85	781
11		25	396	119	1392
12		28	748	161	2494
13		39	1024	215	4442
14		44	1893	293	7854
15		40	2512	396	13899
16		29	4485	506	24215
17		21	5638	632	41802
18		18	9529	788	71167
19		12	10878	985	119888
20		6	16993	1194	198363
21		1	17110	1414	323206
22			23952	1664	515778
23			20224	1884	811000
24			24047	1999	1248011
25			15578	1958	1885279
26			14560	1770	2782396

TABLE I. (Continued)

Depth	2×2	2×3	3×3	2×4	3×4
27			6274	1463	4009722
28			3910	1076	5621354
29			760	667	7647872
30			221	361	10065800
31			2	190	12760413
32				88	15570786
33				39	18171606
34				19	20299876
35				7	21587248
36				1	21841159
37					20906905
38					18899357
39					16058335
40					12772603
41					9515217
42					6583181
43					4242753
44					2503873
45					1350268
46					643245
47					270303
48					92311
49					27116
50					5390
51					1115
52					86
53					18

Weaker variants of 2×3 and 3×3 puzzles were solved in [18]. Furthermore, in that research the empty square of the goal position was not in a corner. The 3×3 case, also called the *8-puzzle*, has recently also been solved in [16] (there is a misprint in [16, Figure 1]—the number of states on level 12 is 748, not 749).

5. The 15-Puzzle and Beyond

Until recently, the first unsolved case was the 15-puzzle. A complete solution of the 15-puzzle by a computer search using the approach in the previous sections requires an amount of memory that is not available in

contemporary computers. For this and larger puzzles, we can try to find good upper and lower bounds on δ. There are several methods for obtaining such bounds, some of which will be discussed here.

5.1. Lower Bounds

Methods have been developed in artificial intelligence to find the shortest distance from a position to the goal position. Any such distance is then a lower bound on δ. We do not discuss the search methods here, but refer the reader to [5].

For large problems, good lower bounds are then obtained by making good guesses about which positions are difficult. Methods have also been developed to find sets of difficult positions [5]. A good approach is usually to reflect the goal position in the center of the puzzle, which for the 15-puzzle leads to the following position:

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

This is certainly a good guess, since it is at distance 78 from the goal position, and for the 15-puzzle we have that $\delta = 80$, proved in [2]. Thirteen positions with distance 80 to the goal position are presented in [5]. Three such positions are given below:

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

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

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

For the 24-puzzle and even larger puzzles, the search methods are in most cases not able to find the exact distance to the goal position within reasonable time. For such cases, the more CPU time is used, the better lower bound on the exact distance we get.

5.2. Upper Bounds

An upper bound is obtained by giving a method that guarantees that the puzzle is solved within a given number of moves, whatever the position is.

This can be done in a way that humans solve these puzzles, which is usually a row-by-row manner.

The 15-puzzle, for example, can be solved in the following way, see [5]. First, the following puzzle is solved. The puzzle contains indistinguishable blocks, which are marked I, and it has $16!/8! = 518918400$ possible states.

1	2	3	4
5	I		I
9	I	I	I
13	I	I	I

We now obtain an upper bound for the 15-puzzle. Namely, the above-mentioned position can be obtained in $\delta = 62$ moves, and thereafter we can solve an 8-puzzle in at most 31 moves to get a total of 93 moves. To get an even better upper bound, however, we calculate the maximum distance to the goal position from any position with the empty square in a given initial position. These distances, in the corresponding positions of the empty space, are

30	29	30
29	30	31
30	31	30

Hence, for the 15-puzzle, $\delta \leq 62 + 29 = 91$. This idea has been further developed in [5] to lower the record down to 87. Recently, Marzetta *et al.* [2] announced a further improvement to 80 using more sophisticated methods. Since this upper bound equals the best known lower bound mentioned earlier, the problem of determining δ has hence been settled.

We end the discussion of upper bounds by presenting a new upper bound for the 24-puzzle.

Theorem 1 *For the 24-puzzle,* $\delta \leq 210$.

Proof: The approach is very much the same as the approach for the 15-puzzle, outlined above. Now, however, the bound is obtained in three steps, rather than in two. We go in the "wrong" direction. The 3×4 puzzle, depicted earlier, has the following maximum distances to the goal position with the empty square in different initial positions:

53	52	51	52
52	51	52	51
53	52	51	52

This table is used to find a good place for the empty square in the goal position of the next puzzle:

	7	8	9	10
11	I	I		I
16	I	I	I	I
21	I	I	I	I

This puzzle has $\delta = 75$, but, again, we do not use this value, but calculate maximum distances to the goal position with the empty square in different initial places:

	75	74	73	74
75	74	75	74	73
74	75	74	73	74
75	74	75	74	75

Finally, we solve the following puzzle:

1	2	3	4	5
6	I	I		I
I	I	I	I	I
I	I	I	I	I
I	I	I	I	I

This puzzle has $\delta = 86$, which means that for the 24-puzzle, $\delta \leq 86 + 73 + 51 = 210$. □

The last two intermediate puzzles in the proof of Theorem 1 have 3047466240 and 2422728000 possible states, respectively. The earlier best known bound was 219, proved in [5].

Acknowledgments

The authors thank Ralph Gasser and Aapo Rautiainen for rewarding discussions. The Center for Scientific Computing (CSC), Espoo, Finland, is acknowledged for providing computing resources.

References

[1] E. R. Berlekamp, J. H. Conway, and R. K. Guy, "Winning Ways for Your Mathematical Plays; Vol. 2: Games in Particular," Academic Press, London, 1982.

[2] A. Brüngger, A. Marzetta, K. Fukuda, and J. Nievergelt, The parallel search bench ZRAM and its applications, *Ann. Oper. Res.*, to appear.

[3] J. C. Culberson and J. Schaeffer, "Efficiently Searching the 15-Puzzle," Technical Report TR 94-08, University of Alberta, Canada, 1994.

[4] A. H. Frey, Jr. and D. Singmaster, "Handbook of Cubik Math," Enslow, Hillside, NJ, 1982.

[5] R. U. Gasser, "Harnessing Computational Resources for Efficient Exhaustive Search," Ph.D. Thesis, ETH, Zürich, 1995.

[6] L. J. Gerstein, "Discrete Mathematics and Algebraic Structures," Freeman, New York, 1987.

[7] O. Hansson, A. Mayer, and M. Yung, Criticizing solutions to relaxed models yields powerful admissible heuristics, *Inform. Sci.* **63** (1992), 207–227.

[8] E. Horowitz and S. Sahni, "Fundamentals of Computer Algorithms," Computer Science Press, Rockville, MD, 1978.

[9] W. W. Johnson and W. E. Storey, Notes on the "15" puzzle, *Amer. J. Math.* **2** (1879), 397–404.

[10] R. E. Korf, Depth-first iterative deepening: An optimal admissible tree search, *Artif. Intel.* **27** (1985), 97–109.

[11] R. E. Korf, Linear-space best-first search, *Artif. Intel.* **62** (1993), 41–87.

[12] S. R. Lerman, "Problem Solving and Computation for Scientists and Engineers: An Introduction Using C," Prentice-Hall, Englewood Cliffs, NJ, 1993.

[13] E. P. M. van Liempd, "Limited-Search Algorithms," Report TI-IR-93-1875, PTT Research Groningen, The Netherlands, 1993.

[14] I. Parberry, A real-time algorithm for the $(n^2 - 1)$-puzzle, *Inform. Process. Lett.* **56** (1995), 23–28.

[15] D. Ratner and M. Warmuth, Finding a shortest solution for the ($N \times N$)-extension of the 15-puzzle is intractable, *J. Symbolic Comput.* **10** (1990), 111–137.

[16] A. Reinefeld, Complete solution of the eight-puzzle and the benefit of node ordering in IDA*, in: "IJCAI-93: Proceedings of the 13th International Joint Conference on Artificial Intelligence," Morgan Kaufmann, San Mateo, CA, 1993, pp. 248–253.

[17] E. M. Reingold, J. Nievergelt, and N. Deo, "Combinatorial Algorithms: Theory and Practice," Prentice-Hall, Englewood Cliffs, NJ, 1977.

[18] P. D. A. Schofield, Complete solution of the "eight-puzzle", in: N. L. Collins and D. Mitchie (eds.), "Machine Intelligence 1," Oliver & Boyd, Edinburgh, 1967, pp. 125–134.

[19] P. G. Tait, Note on the theory of the "15" puzzle, *Proc. Roy. Soc. Edinburgh* **10** (1880), 664–665.

[20] R. M. Wilson, Graph puzzles, homotopy and the alternating group, *J. Combin. Theory Ser. B* **16** (1974), 86–96.