Pushing Blocks is NP-Complete for Noncrossing Solution Paths

Erik D. Demaine* Michael Hoffmann†

Abstract

We prove NP-hardness of a wide class of pushing-block puzzles like the classic Sokoban, generalizing several previous results [4, 5, 7, 8, 13, 15]. The puzzles consist of unit square blocks on an integer lattice; all blocks are movable. The robot may move horizontally and vertically in order to reach a specified goal position. In the PUSH-k puzzle, the robot can push up to k blocks in front of it as long as there is at least one free square ahead. Other variations were introduced to make puzzles more tractable, in which blocks must slide their maximal extent when pushed (PUSH-PUSH), and in which the robot's path must not cross itself (PUSH-X). We prove that all of these puzzles are NP-hard.

Keywords: Motion planning, combinatorial games, computational complexity.

1 Introduction

Algorithmic motion planning is a large area of computational geometry with applications in robotics, assembly planning, and computer animation; see, e.g., [14] for a survey. The standard type of problem involves moving a robot from one configuration to another while avoiding fixed obstacles. A recent direction introduced by Wilfong [15] is a class of problems in which robots are permitted to move some of the obstacles in order to increase maneuverability. As robots become more powerful at manipulation, an understanding of such models becomes increasingly important. Current-day applications include automated warehouse control and warehouse navigation; see, e.g., [9]. A representative abstraction of such applications is the popular Sokoban puzzle [3, 8], which is known to be PSPACE-complete [3]. In this paper we study several variations of simpler puzzles, and show all of these models are NP-hard using a reductions from 3-coloring of planar graphs [10]. Some variations are additionally known to be NP-complete, others PSPACE-complete, while the complexity of most variations is unresolved between NP and PSPACE.

*Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, eddemaine@uwaterloo.ca.

†Institute for Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland, hoffmann@inf.ethz.ch.

Problems. Our hardness results are particularly surprising given the simplicity of the model of motion and obstacle manipulation. Consider a rectangular $n \times m$-grid in which each square is marked either *free* or *blocked*. A *robot* can move horizontally and vertically in the grid, and thereby push up to k blocks in front of it, for some constant k. See Figure 1, in which the blocked positions are shaded and the robot is shown as circle, pushing two blocks.

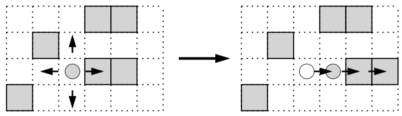


Figure 1: Example of pushing blocks.

The PUSH-k problem [4, 5] is to decide whether there is a sequence of moves starting at a specified free position and ending at a specified goal position. If we omit the restriction on how many blocks the robot can push at once (i.e., $k = \infty$), we obtain the problem PUSH-∗ [2, 6, 7, 12].

To make these basic problems more computationally tractable, we can impose additional simplifying constraints on the robot's motion. The PUSH-PUSH model [4, 5, 13] requires that, once a block is pushed, it slides the maximal extent in that direction. This model can be thought of representing either sliding blocks on a frictionless surface, or the situation in which blocks cannot be pushed by precise amounts but can be consistently pushed against other blocks. The PUSH-X model [4] requires that the robot not cross its own path. This model is not intended to represent reality, but rather was proposed to severely restrict the motions in order to make the problem computationally tractable. In particular, any version of PUSH-X is automatically in NP, unlike the general problem.

Related Work. Out of these many problem variations, several individual cases have been studied. The original paper by Wilfong [15] studies a more flexible model in which the blocks can be more general than squares, and the robot can both push and pull blocks. Dhagat and O'Rourke [7] initiated the PUSH- line of models, and proved that PUSH-∗ is NP-hard if some blocks can be tied to the board, making them unpushable. This result was later strengthened to PSPACE-completeness [2].

Our models disallow blocks from being tied to the board, making the problem easier to solve and hardness gadgets more difficult to construct. Progress in this setting has only been made in the last two years, during which NP-hardness was proved for Push-Push-1 [4, 5, 13], Push-1 [4], and Push-* [12]. These results leave two aspects open: Push-k for general k, and the Push-X noncrossing restriction. None of the previous reductions extend easily to these scenarios, except for the Push-* reduction [12] which also applies to Push-*-X.

Results. In this paper, we provide one construction that settles the NP-hardness of Push-k, Push-k-X, Push-Push-k, and Push-Push-k-X, for any fixed $k \in \mathbb{N}$. In particular, our results subsume a number of previous NP-hardness proofs of pushing-block variations [4, 5, 7, 8, 13, 15], and solves the Push-X open problem from [4]. Thus, this paper together with [12] prove that all pushing-block puzzles described above are NP-hard. The new idea in our reduction is to force the robot to follow constrained Eulerian tours of planar graphs and carry a constant amount of information along each edge of the graph. This idea is in contrast to all previous approaches of building circuits based on graphs, which seem to inherently require crossings.

2 The Reduction

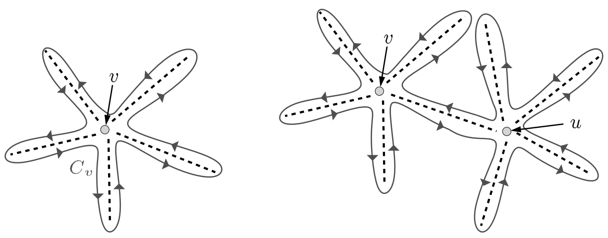
We first consider the problem Push-1-X, in which the robot is restricted to push only one block at any time. Later we will describe how to modify the construction for $k > 1$.

Our reduction is from planar 3-coloring, which is known to be NP-complete, even if no vertex has degree larger than four [10]. We are given a planar embedding of a connected undirected graph $G = (V, E)$, and will construct a Push-1 puzzle from G that is solvable if and only if G is (vertex) 3-colorable.

Let $\overrightarrow{G}$ be the directed planar graph resulting from G by replacing each undirected edge by two directed edges of opposite orientation. By a well-known theorem (cf. [11]), there is a Eulerian tour in $\overrightarrow{G}$, since for every vertex the number of incoming edges equals the number of outgoing edges. Moreover, we claim that there is always a *planar* Eulerian tour T in $\overrightarrow{G}$, where planar means that it does not even cross itself at vertices, i.e., can be drawn with a pencil in one piece without crossing. (The fact that edges do not intersect in their interior is already implied by the planarity of G.)

Lemma 1 (See also [1].) *There is a planar Eulerian tour T in $\overrightarrow{G} = (V, \overrightarrow{E})$.*

Proof. Consider an arbitrary vertex $v \in V$ and the cycle C_v visiting all edges from $\overrightarrow{E}$ adjacent to v in counterclockwise order as shown in Figure 2(a). Now start a breadth-first traversal of $\overrightarrow{G}$ from v. Let $u \in V$ be the next vertex visited in this traversal. Then the cycles C_v and C_u can be joined by overlaying and cutting at their common edges, $\overrightarrow{uv}$ and $\overrightarrow{vu}$ as shown in Figure 2(b). No other edge of C_u can be in conflict with an edge from the tour constructed so far, as then u would have been visited before. After all vertices have been visited, the result is a planar Eulerian tour of $\overrightarrow{G}$. $\square$



(a) The cycle C_v. (b) Joining cycles.

Figure 2: Constructing a planar Eulerian tour.

The idea is to use T for traversing $\overrightarrow{G}$, thereby

- choosing the color of a vertex whenever leaving it and
- checking that the adjacent vertices are colored differently whenever traversing the second of the both edges from $\overrightarrow{G}$ representing an edge from G.

Of course, we must ensure that the colors chosen for the vertices are consistent, i.e., anytime we leave a specific vertex, the same color has to be chosen.

For this, we let two (directed) edges leaving the same vertex meet in what we call a *consistency gadget*, in order to assure that the same color is chosen in both. Of course, it might not be possible to join any pair of edges leaving a specific vertex this way without crossings, but it is sufficient to join the edges such that they form a tree under those junctions, and then all edges have to choose the same color by transitivity. Refer to the example in Figure 3: the edges of the graph G are drawn as dashed lines, the solid curve is a planar Eulerian tour of $\overrightarrow{G}$, and the wiggled segments denote the consistency junctions.

Similarly, any pair of edges representing the same edge of the original graph G are joined; these *coloring junctions* are shown as solid arrows in Figure 3. It is clear how to draw them, since both edges are always adjacent in the drawing. Also, coloring and consistency junctions do not interfere, since all of the latter are close to a vertex. By placing an appropriate gadget on the coloring junctions, we can forbid to choose the same color for adjacent vertices.

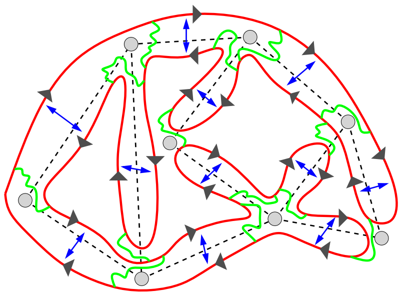


Figure 3: Example graph, tour and junctions.

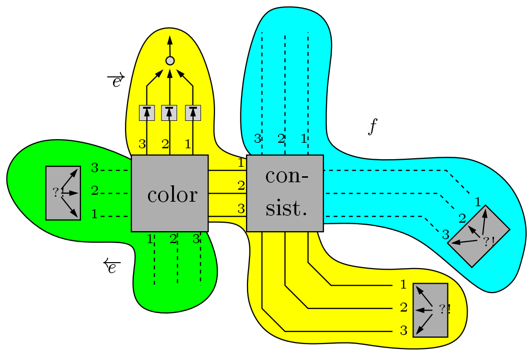


Figure 4: Construction for an edge.

The construction uses several gadgets, as described in Table 1. In the left column, we give a symbol to denote the type of gadget; in the right column we list the gadget's name, in parentheses its number of entries and exits, and a short description of its functionality. These gadgets are explained in detail in Sections 2.3–2.6.

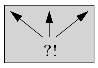
	One-way gadget (1/1); can only be passed in direction of the arrow. However, once this happened, it is open, i.e. can be traversed in both directions arbitrarily.
	Fork gadget (1/3); can be left through any of the three exits, but only one of these paths may be entered.
	XOR-crossing gadget (2/2); a safe, leakage free crossing, provided that only one of the two paths is actually traversed.
	NAND gadget (2/2); joins two paths such that at most one of them can be traversed.

Table 1: The gadgets.

Since the robot can push at most one block, any two-times-two square of blocks is essentially fixed. Hence, one can build the Eulerian tour as a PUSH-1 puzzle, using corridors that are surrounded by walls of thickness at least two. More precisely, each edge $\overrightarrow{e} = (u, v)$ from $\overrightarrow{G}$ will be represented by three corridors in the puzzle, as shown in Figure 4; the robot can choose to traverse any of them (but only one) through a fork gadget, this way assigning a color to the vertex u it leaves. The corridors of $\overrightarrow{e}$ are joined to the corridors of another edge f also leaving u (if existent) through a consistency gadget which guarantees that only the corridors of matching color can be traversed. Similarly, the corridors are joined to those corresponding to the opposite edge $\overleftarrow{e} := (v, u)$ through a coloring gadget, making sure that u and v are colored differently. Finally, the three paths rejoin, protected by one-way gadgets preventing the robot from stumbling backwards in the wrong direction.

2.1 The Coloring Gadget

This gadget is used to make sure that adjacent vertices do not get the same color. It joins two paths of three labeled corridors each in a symmetric way such that corridors with the same label are joined into a NAND gadget; see Figure 5.

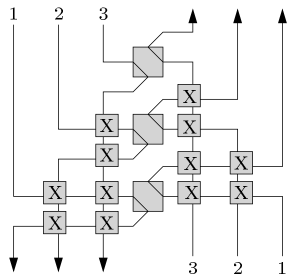


Figure 5: Coloring gadget.

Lemma 2 *Both sides of a coloring gadget can be traversed if and only if the robot chooses differently labeled paths.*

Proof. Note that entrance and exit to the gadget are guarded by a fork gadget, one-way gadget, respectively (see Figure 4). Hence, at most one of the corridors on each side can be traversed and the XOR-crossings are safe. □

2.2 The Consistency-Check Gadget

This gadget is complementary to the coloring gadget. It is used to make sure that the robot cannot choose different colors for the same vertex, that is anytime it leaves the vertex, the same color has to be chosen. The gadget joins two paths of three labeled corridors each in a symmetric way such that corridors with the different labels are joined into a NAND gadget; see Figure 6.

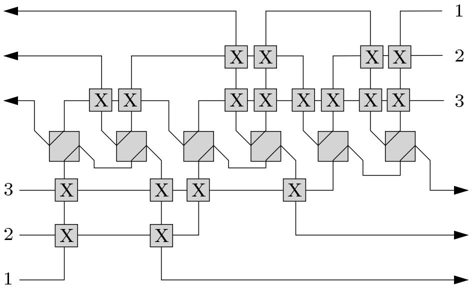


Figure 6: Consistency Gadget.

Lemma 3 *Both sides of a consistency gadget can be traversed if and only if the robot chooses paths with the same label.*

Proof. As in Lemma 2. □

2.3 Nand Gadget

To make sure that only specific combinations of corridors are used by the robot, we need a gadget joining two corridors in such a way that at most one of them can be traversed: a Nand gadget. The layout is shown in Figure 7; it depends on whether both corridors are to be traversed in the same or in opposite directions.

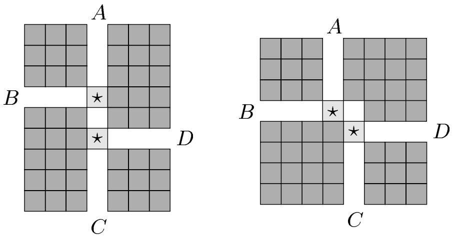


Figure 7: Nand gadgets.

Lemma 4 *Consider the gadgets shown in Figure 7.*

The left gadget can be traversed either $A \to B$ or $C \to D$ but not both ways.

The right gadget can be traversed either $B \to A$ or $C \to D$ but not both ways.

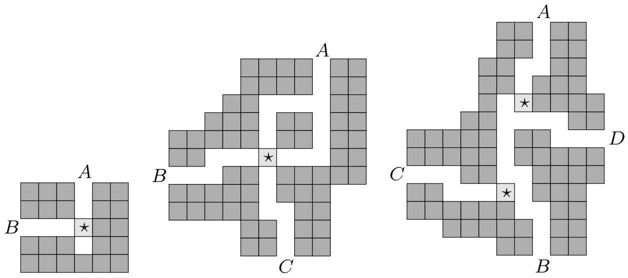
Proof. In order to traverse the gadget, one of the blocks marked with $\star$ has to be pushed. Wherever this is done, afterwards the two marked blocks are lined up in sequence, making them un-pushable in direction of their lineup. (Remember that the robot cannot push more than one block at once.) Since they can neither be pushed apart from the side, there is no way to move the marked block that has not been pushed so far; but this would be necessary to traverse the gadget on the other way. □

Remark 1 *For Push-k each marked block in the left gadget would have to be replaced by a sequence of $\lceil \frac{k+1}{2} \rceil$ blocks.*

Since it is always clear from the Eulerian tour in which direction an edge is to be traversed, there is no problem in choosing the appropriate type of Nand-gadget. Hence, we refer to this class of gadgets by the term "Nand-gadget" as if there was just a single one.

2.4 One-Way Gadget

The one-way shown in Figure 8(a) can be traversed in only one direction, from A to B. But note that once it has been traversed, it is just an open corridor that can be traversed in both directions.



(a) One-Way (b) Fork (c) Xor-Crossing

Figure 8: The basic gadgets (after Figs. 2, 3, 10 of [4]).

2.5 Fork Gadget

The fork gadget shown in Figure 8(b) allows traversal from A to B or C, but once either of B or C has been reached, the other is inaccessible from A.

Note that it is possible to go e.g. from A to B and then from A to C, if the blocking square is pushed out of the corridor to C in between. But in our construction this can never occur, since the exits of the forked corridors are protected by one-way gadgets.

Also note that a three-way fork as in Table 1 is just a combination of two two-way forks.

2.6 Xor-Crossing Gadget

The Xor-crossing shown in Figure 8(c) can be traversed either from A to B or from C to D without leakage, i.e. it cannot be traversed from A to C or D or from C to A or B.

Note that the above statement is true for a single traversal only or more precisely, if the gadget is traversed only from either of A or C. This condition is fulfilled wherever Xor-crossings are used in our construction, as already discussed in Lemma 2.

2.7 Main Theorems

In order to actually define the Push-1 puzzle, we need a start and a goal position for the robot. This is easily done by breaking the tour at some arbitrary vertex, yielding a path with start and goal position at its ends.

Theorem 5 Push-*1 is NP-hard.*

Proof. For the given planar graph $G = (V, E)$ construct the Push-1 puzzle as described above. The size of the puzzle is determined by the number of gadgets and the number of gadgets of each single type is linear in the number of edges in G. Hence, the size of the puzzle is polynomial in the size of G.

If the robot finds a way to the goal position, by Lemma 3 each time it leaves a vertex it chooses the corridor corresponding to the same color. This defines a mapping $c : V \to \{1, 2, 3\}$. Moreover, by Lemma 2, this mapping is a coloring.

If on the other hand there is a 3-coloring $c : V \to \{1, 2, 3\}$ of G, then the robot can find a way to the goal position by consistently following at each vertex v the corridor corresponding to $c(v)$. □

Theorem 6 Push-k *is NP-hard for any fixed* $k \in \mathbb{N}$.

Proof. As in Theorem 5. But corridors have to separated by walls of at least $k + 1$ blocks and the Nand-gadget has to adopted as indicated in Remark 1. □

Corollary 7

- Push-k-`X` *is NP-hard.*
- Push-Push-k *is NP-hard.*
- Push-Push-k-`X` *is NP-hard.*

Proof. There is always a path for the robot that does not cross itself and wherever blocks are pushed, they are pushed as far as possible. □

3 Conclusion

We have shown NP-hardness of a broad class of pushing-block puzzles. Except for the noncrossing Push-`X` variants, it remains open whether these problem are in NP. Some of the problems might be PSPACE-complete, as has been shown for the fixed-block version of Push-`*` [2].

Another still open question is whether there is an "interesting" puzzle variant that is still tractable. Up to now, the only problem known to be in P is if the robot is restricted to monotonic paths [7], e.g., it can only move up and right.

Acknowledgments

We thank Martin Demaine and Joseph O'Rourke for helpful discussions.

References

[1] Biedl, T., Kaufmann, M., and Mutzel, P. Drawing planar partitions ii: Hh-drawings. In *Proc. 24th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.* (1998), vol. 1517 of *Lecture Notes Comput. Sci.*, Springer-Verlag, pp. 124–136.

[2] Bremner, D., O'Rourke, J., and Shermer, T. Motion planning amidst movable square blocks is PSPACE complete. Draft, 1994.

[3] Culberson, J. Sokoban is PSPACE-complete. In *Proc. Internat. Conf. Fun with Algorithms* (Elba, Italy, June 1998), N. S. E. Lodi, L. Pagli, Ed., Carelton Scientific, pp. 65–76.

[4] Demaine, E. D., Demaine, M. L., and O'Rourke, J. PushPush and Push-1 are NP-hard in 2D. In *Proc. 12th Canad. Conf. Comput. Geom.* (2000), pp. 211–219.

[5] Demaine, E. D., Demaine, M. L., and O'Rourke, J. PushPush is NP-hard in 2D. Technical Report 065, Dept. Comput. Sci., Smith College, Northampton, MA, Jan. 2000.

[6] Demaine, E. D., and O'Rourke, J. Open problems from CCCG'99. Technical Report 066, Dept. Comput. Sci., Smith College, Northampton, MA, Mar. 2000.

[7] Dhagat, A., and O'Rourke, J. Motion planning amidst movable square blocks. In *Proc. 4th Canad. Conf. Comput. Geom.* (1992), pp. 188–191.

[8] Dor, D., and Zwick, U. Sokoban and other motion planning problems. *Computational Geometry: Theory and Applications 13*, 4 (1999), 215–228.

[9] Everett, H. R., and Gage, D. W. From laboratory to warehouse: Security robots meet the real world. *Internat. J. Robotics Research 18*, 7 (July 1999), 760–768.

[10] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[11] Harary, F. *Graph Theory.* Addison-Wesley, Reading, MA, 1972.

[12] Hoffmann, M. Push-`*` is NP-hard. In *Proc. 12th Canad. Conf. Comput. Geom.* (2000), pp. 205–210.

[13] O'Rourke, J., and the Smith Problem Solving Group. PushPush is NP-hard in 3D. Technical Report 064, Dept. Comput. Sci., Smith College, Northampton, MA, Nov. 1999.

[14] Sharir, M. Algorithmic motion planning. In *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. CRC Press LLC, Boca Raton, FL, 1997, ch. 40, pp. 733–754.

[15] Wilfong, G. Motion planning in the presence of movable obstacles. *Ann. Math. Artif. Intell. 3* (1991), 131–150.