Pushing Blocks is Hard

Erik D. Demaine [a], Martin L. Demaine [a],
Michael Hoffmann [b] and Joseph O'Rourke [c,1]

[a] *MIT Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA.*

[b] *Institute for Theoretical Computer Science, ETH Zürich, CH-8092 Zürich, Switzerland.*

[c] *Department of Computer Science, Smith College, Northampton, MA 01063, USA.*

Abstract

We prove NP-hardness of a wide class of pushing-block puzzles similar to the classic Sokoban, generalizing several previous results [5,6,9,10,15,17]. 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. The puzzle variants differ in the number of blocks that the robot can push at once, ranging from at most one (Push-1) up to arbitrarily many (Push-*). Other variations were introduced to make puzzles more tractable, in which blocks must slide their maximal extent when pushed (PushPush), and in which the robot's path must not revisit itself (Push-X). We prove that all of these puzzles are NP-hard.

Key words: 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., [16] 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 [17] is a class of problems in which robots are permitted

Email addresses: `edemaine@mit.edu` (Erik D. Demaine), `mdemaine@mit.edu` (Martin L. Demaine), `hoffmann@inf.ethz.ch` (Michael Hoffmann), `orourke@cs.smith.edu` (Joseph O'Rourke).

[1] Supported by NSF Distinguished Teaching Scholars award DUE-0123154.

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., [11]. A representative abstraction of such applications is the popular Sokoban puzzle [4,10], which is known to be PSPACE-complete [4]. In this paper we study several variations of simpler puzzles, and show all of these models are NP-hard. Some variations are additionally known to be NP-complete, others PSPACE-complete, while the complexity of most variations is unresolved between NP and PSPACE.

1.1 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 a circle, pushing two blocks.

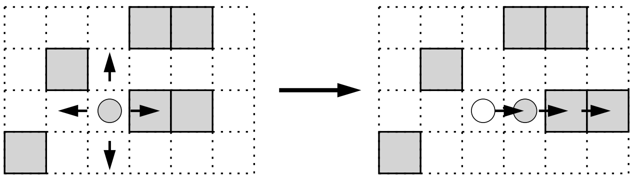


Fig. 1. Example of pushing blocks.

The PUSH-k problem [5,6] is to decide whether there is a sequence of moves starting at a specified free position and ending at a specified goal position. Note that as a significant difference compared to Sokoban, we may not specify storage locations for the obstacle blocks.

If we omit the restriction on how many blocks the robot can push at once (i.e., $k = \infty$), we obtain the problem PUSH-$*$ [3,8,9,14]. Here it is necessary to bound the arena of action (the "board") with an unpushable rectangular boundary, as otherwise the robot can reach any goal.

To make these basic problems more computationally tractable, we can impose additional simplifying constraints on the robot's motion. The PUSHPUSH model [5,6,15] requires that, once a block is pushed, it slides the maximal extent in that direction, i.e., until it hits another block or the outer boundary. 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. In analogy to the PUSH- terminology, we append the maximal number of blocks that can be pushed at the same time, such that, e.g., PushPush-1 refers to the PushPush model, where at most one block can be pushed at any time.

The Push-X and PushPush-X models [5] require that the robot does not revisit its own path, i.e., it is allowed to be in each square at most once. Although one could contrive situations in which this model might apply (and it does appear in several games, e.g., Chip's Challenge), it was proposed primarily to restrict the motions in order to make the problem computationally tractable. In particular, any version of Push-X or PushPush-X is automatically in NP, unlike the general problem.

1.2 Related Work

Out of these many problem variations, several individual cases have been studied. The original paper by Wilfong [17] 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 [9] 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 [3].

Our models disallow blocks from being tied to the board, making the problem easier to solve and hardness proofs more challenging. Progress in this setting has been made only in the last two years, during which NP-hardness was proved for PushPush-1 [5,6,15] and Push-1 [5]. These results leave three classes of problems open: Push-k for general k, Push-*, and the Push-X noncrossing restriction. None of the previous reductions extend easily to these scenarios.

1.3 Results

In this paper, we provide two constructions that together prove the NP-hardness of all pushing-block puzzles described above. The first reduction applies to Push-k, Push-k-X, PushPush-k, and PushPush-k-X, for any fixed $k \in \mathbb{N}$. The second reduction applies to Push-* and Push-*-X (and to their PushPush variants). In particular, our results subsume a number of previous NP-hardness proofs of pushing-block variations [5,6,9,10,15,17], and solve the Push-X open problem from [5].

The new idea in our first 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. On the other hand, our second reduction employs a small amount of carefully positioned empty space to build a kind of logic engine out of a SAT matrix. Both ideas are in contrast to all previous approaches of building circuits based on graphs, which seem to inherently require problematic crossings.

Preliminary versions of each reduction have appeared in [7] and [14].

2 Push-k-X

We first consider the problem Push-1-X, in which the robot is restricted to push only one block at any time, and the robot may not revisit its own path. Later we will describe how to modify the construction for $k > 1$.

Our reduction is from planar 3-coloring. In this problem, one is given a planar embedding of a connected undirected graph $G = (V, E)$, and one has to decide whether or not this graph is (vertex) 3-colorable. A (vertex) 3-coloring for G is a mapping $c : V \to \{1, 2, 3\}$, such that $c(u) \neq c(v)$ for all edges $(u, v) \in E$. A graph is said to be 3-colorable, if and only if there exists a 3-coloring for it. The planar 3-coloring problem is known to be NP-complete, even if no vertex has degree larger than four [12]. In the reduction, we will construct a Push-1 puzzle for the given graph G (in polynomial time) that is solvable if and only if G is (vertex) 3-colorable.

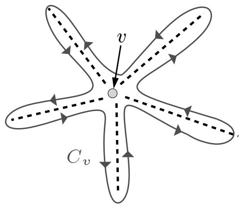
Let $\overrightarrow{G}$ be the directed planar graph resulting from G by replacing each undirected edge by two directed edges of opposite orientation. By a well-known theorem (cf. [13]), there is a Eulerian tour in $\overrightarrow{G}$, because for every vertex the number of incoming edges equals the 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 *There is a planar Eulerian tour T in $\overrightarrow{G} = (V, \overrightarrow{E})$.*

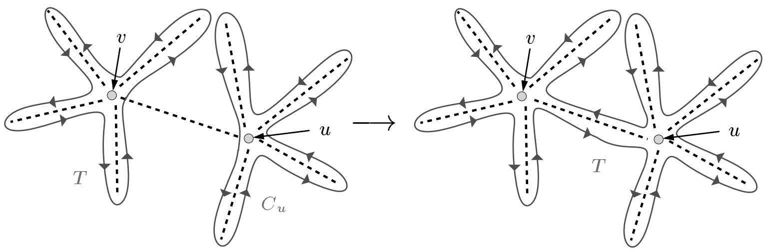
Proof. Start a breadth-first traversal of $\overrightarrow{G}$ from some arbitrary vertex v_0. For a vertex $v \in V$ define $U(v) \subset V$ to be the set of not-yet-visited neighbors of v. Let C_v be the cycle visiting all edges of the subgraph induced by $\{v\} \cup U(v)$ in counterclockwise order (Figure 2(a)).

We build the tour T in an incremental manner, starting with C_{v_0}: at each step we include all edges which are not yet part of T and which are incident to the visited vertex. Assume the breadth-first traversal next visits vertex u, and let v be the ancestor vertex of u in the traversal, i.e., the vertex that caused u to be visited. Then join T and C_u by cutting them at v and u, respectively, where they cross the edge vu, and connecting the resulting paths using the edges $\overrightarrow{vu}$ and $\overrightarrow{uv}$ (Figure 2(b)). Since all cycles C_v, $v \in V$ are oriented counterclockwise, and since T is built from such cycles only, the orientations of C_u and T in the join step match. Thus, after all vertices have been processed, the resulting tour T is a planar Eulerian tour of $\overrightarrow{G}$. □

Another proof for Lemma 1 can be found in [2].



(a) A cycle C_v.



(b) Extending the tour.

Fig. 2. Constructing a planar Eulerian tour.

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

(1) choosing the color of a vertex whenever leaving it and
(2) checking that the adjacent vertices are colored differently whenever traversing the second of the two edges of $\overrightarrow{G}$ representing an edge of G.

Of course, we must ensure that the colors chosen for the vertices are consistent, i.e., each time we leave a specific vertex, the same color is transmitted along the outgoing edge.

To achieve this consistency property, 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 every 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; see Figure 3. In fact, since the maximum vertex degree in G is four, it can easily be verified that we can replace "tree" by "path" in the above statement.

Similarly, we join every pair of directed edges representing the same undirected edge of the original graph G with *coloring junctions* (cf. Figure 3). It is clear how to draw these without crossings, because the two directed edges are adjacent in the drawing. Also, coloring and consistency junctions do not interfere, because consistency junctions are near a vertex. By placing an appropriate gadget on the coloring junctions, we can forbid adjacent vertices from choosing the same color.

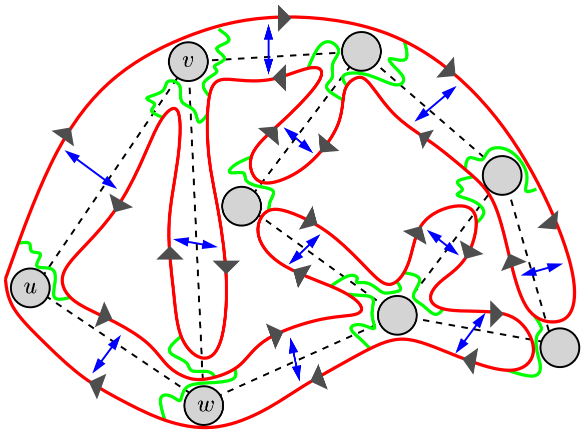
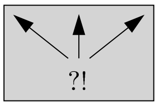
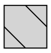


Fig. 3. Example graph (dashed lines), planar Eulerian tour (solid curve), and junctions. Consistency junctions are depicted by wiggled segments, and coloring junctions by solid arrows.

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.

Because the robot can push at most one block, any 2×2 square of blocks is essentially fixed. Hence, the robot can be forced to follow a Eulerian tour by constructing a Push-1 puzzle with corridors that are surrounded by walls of thickness at least two. In fact, each edge $\overrightarrow{e} = (u, v)$ of $\overrightarrow{G}$ 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, and in this way assigns a color to the vertex u it leaves. The corridors of $\overrightarrow{e}$ are joined to the corridors of another edge also leaving u (if such edges exist) through a consistency gadget which guarantees that only the corridors of matching color can be traversed. Similarly, the corridors are joined to the corridors corresponding to the opposite edge $\overleftarrow{e} := (v, u)$ through a coloring gadget, ensuring that u and v are colored differently. Finally, the three paths rejoin, protected by one-way gadgets which prevent the robot from stumbling backwards in the wrong direction.

Table 1
The gadgets.

	One-way gadget (1/1); can only be traversed 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 as soon as one of the three paths is entered, the other two are permanently blocked from this side. If one of the blocked paths can be entered from its other end, the fork can be opened such that all three paths are accessible at the same time. To prevent this, we will usually put one-way gadgets at the other ends of all three paths.
X	Xor-crossing gadget (2/2); a safe, leakage free crossing, provided that only one of the two paths is actually traversed. The robot can, e.g., go from top to bottom, or from left to right, but not from top to right or from left to bottom. However, if both paths (left to right and top to bottom) are traversed, then the robot can reach both exits during the second traversal, and it might turn into the wrong, noncrossing direction (the crossing "leaks").
	Nand gadget (2/2); joins two paths such that at most one of them can be traversed. If one path has been traversed, the other is permanently blocked.

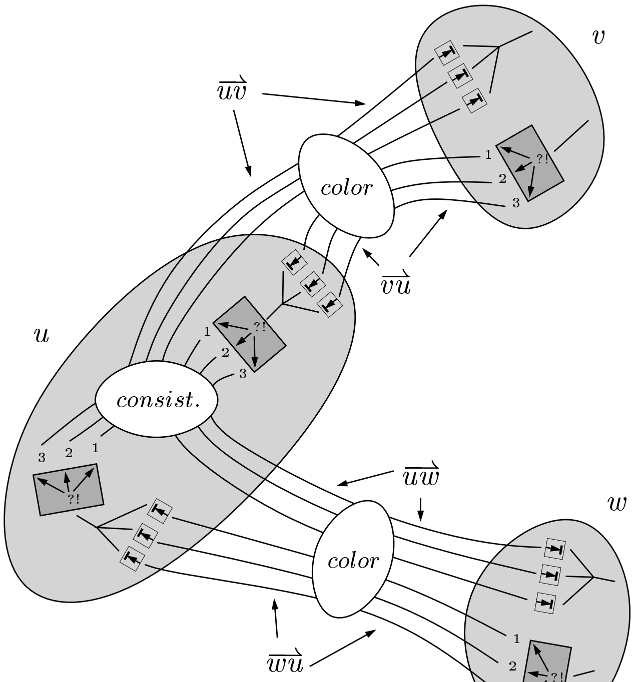


Fig. 4. Construction for the leftmost vertex from Figure 3.

2.1 The Coloring Gadget

This gadget is used to ensure that adjacent vertices are not assigned the same color.

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

Proof. A coloring gadget joins two paths corresponding to directed edges $\overrightarrow{e} = (u, v)$ and $\overleftarrow{e} = (v, u)$ of $\overrightarrow{G}$; each path in turn consists of three corridors labeled 1, 2, and 3, corresponding to the three possible colors for u and v. Since the three corridors of a path are guarded by a fork gadget from one side and by a one-way gadget from the other side (cf. Figure 4), the robot can traverse at most one of the corridors. Hence, the requirements for the XOR crossings (cf. Table 1) used in the coloring gadget are fulfilled.

Since both corridors labeled 1 are joined into a NAND gadget, it is not possible for the robot to traverse both of them, i.e., not both u and v can get color 1. The same argument holds for colors 2 and 3. $\square$

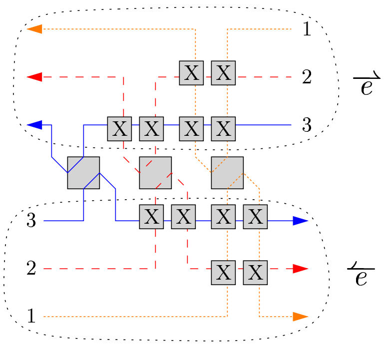


Fig. 5. Coloring Gadget.

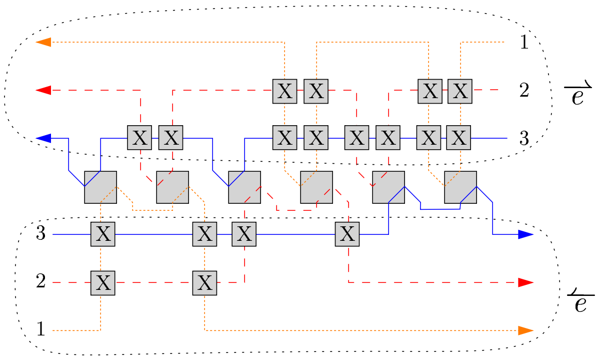


Fig. 6. Consistency Gadget.

2.2 The Consistency-Check Gadget

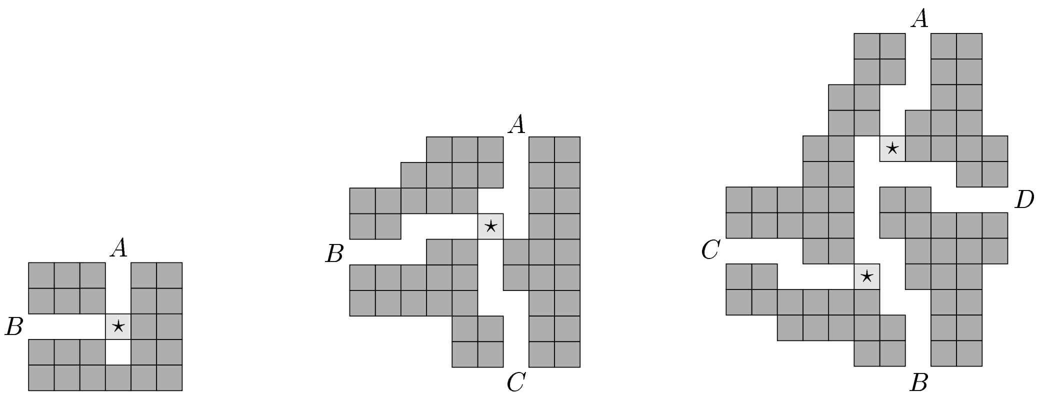
This gadget is complementary to the coloring gadget. It is used to ensure that the robot cannot choose different colors for the same vertex, that is, each time the robot leaves the vertex, it chooses the same color. The gadget joins two paths each consisting of three labeled corridors in a symmetric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

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 One-Way Gadget

The one-way gadget shown in Figure 7(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

Fig. 7. Some basic gadgets from [5] (after Figs. 2, 3, 10 of [5]).

2.4 Fork Gadget

The fork gadget shown in Figure 7(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.

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 scenario can never occur, because the exits of the forked corridors are protected by one-way gadgets.

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

2.5 Xor-Crossing Gadget

The Xor-crossing gadget shown in Figure 7(c) can be traversed either from A to B or from C to D without leakage, that is, the gadget cannot be traversed from A to C or D, or from C to A or B. This statement is true only for a single traversal, or more precisely, if the gadget is traversed only from one of A or C. This condition is fulfilled wherever our construction uses Xor-crossings, as already discussed in Lemma 2.

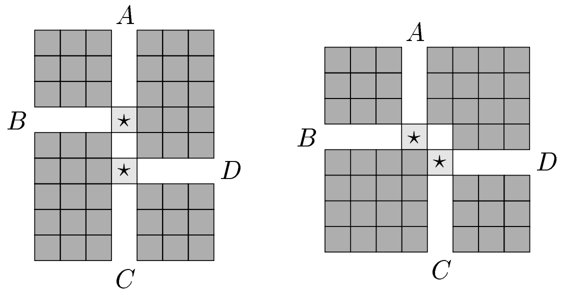


Fig. 8. The Nand gadget.

2.6 Nand Gadget

To ensure 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 8; it depends on whether both corridors are to be traversed in the same or in opposite directions.

Lemma 4 *The left gadget in Figure 8 can be traversed either from A to B or from C to D but not both ways. The right gadget in Figure 8 can be traversed either from B to A or from 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 unpushable by a Push-1 robot in the direction of their alignment. Because the marked blocks also cannot 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 in the other way. $\square$

Remark 5 *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.*

Because the Eulerian tour specifies the direction in which each edge is traversed, there is no problem in choosing the appropriate type of NAND-gadget. Thus we refer to this class of gadgets by the term "NAND-gadget" as if there were just a single one.

2.7 Main Theorems

To complete the definition of 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 6 PUSH-*1 is NP-hard.*

Proof. For a given planar graph $G = (V, E)$, construct the PUSH-1 puzzle as described above. The number of gadgets of each type is linear in the number of edges in G, and each gadget has constant size. By standard results on orthogonal drawings of planar graphs (see, e.g., [1]), the connections between gadgets can be embedded in polynomial area using polynomial time. Thus the puzzle can be constructed in time polynomial in the size of G.

If the robot finds a route to the goal position, then 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 path to the goal position by consistently following at each vertex v the corridor corresponding to $c(v)$. □

Theorem 7 PUSH-k *is NP-hard for any fixed* $k \in \mathbb{N}$.

Proof. As in Theorem 6, except that corridors have to separated by walls of at least $k+1$ blocks, the One-Way, Fork, and XOR-gadgets need the appropriate obvious thickening, and the NAND-gadget has to be adapted as indicated in Remark 5. □

Corollary 8 PUSH-k-X, PUSHPUSH-k, *and* PUSHPUSH-k-X *are 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 Push-*-X

While the construction from Section 2 works for any fixed number of blocks that the robot can push, it does not easily apply to the case in which this number is unbounded. A basic reason for this is that corridors can no longer be separated by walls of sufficiently many blocks. Indeed, what makes Push-* constructions difficult is their nonlocality: gadgets placed arbitrarily far apart still influence each other if they share a common row or column. Hence, we will employ a completely different construction, a reduction from 3-Sat.

In the 3-Sat problem, we are given a boolean formula in conjunctive normal form where each clause consists of exactly three distinct literals. Let $X := \{x_1, \ldots, x_k\}$ be the set of variables and $C_1, \ldots, C_\ell$ the clauses. Define

$$n_i := \left|\left\{r \in \{1, \ldots, k\} \mid x_i \text{ occurs in } C_r\right\}\right|$$

to be the number of occurrences of x_i in the formula. Analogously, let $\overline{n_i}$ be the number of occurrences of $\overline{x_i}$. Without loss of generality, we assume that $n_i \geq \overline{n_i}$ for all i.

We will construct an instance of Push-* such that the puzzle is solvable if and only if there is a satisfying assignment for the given formula. Refer to Figure 9(b) for a complete example of the construction.

The construction consists of four major *blocks* (components), as depicted in Figure 9(a). All positions outside these blocks are initially blocked.

Variables The start position is here and the variable assignment is chosen implicitly when traversing this block. (Section 3.2)

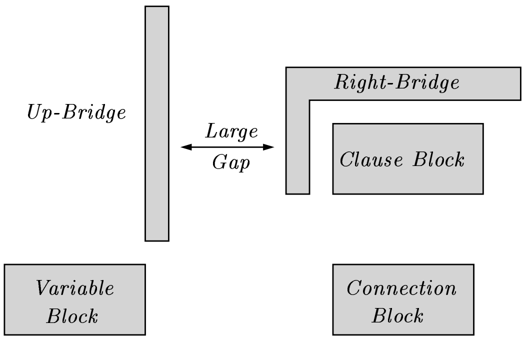
Clauses The goal position is here. It can be reached if and only if the assignment satisfies all clauses. (Section 3.4)

Connections logically connect the variable and clause block. (Section 3.1)

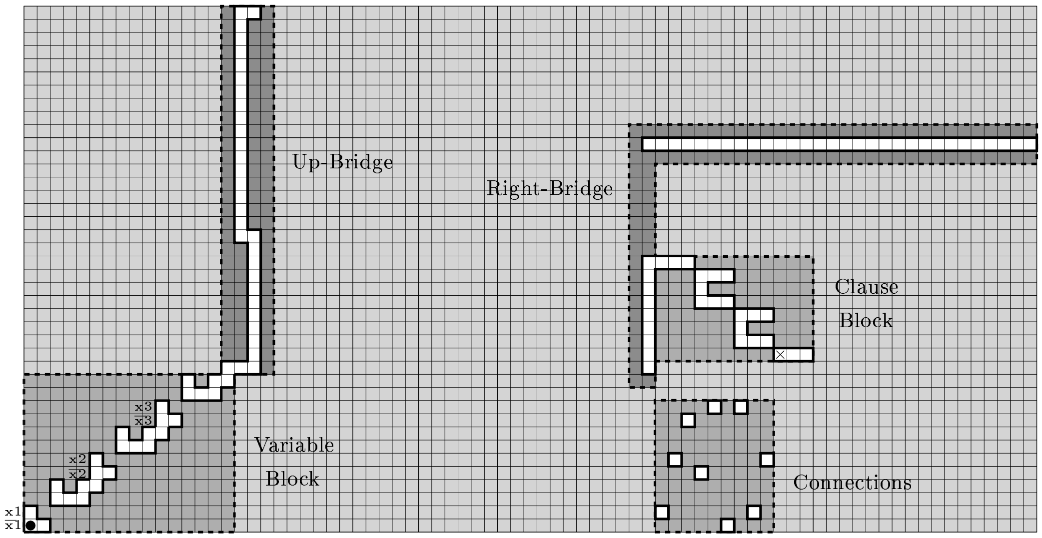
Bridges physically connect (in terms of movement) the variable and clause block. (Section 3.3)

3.1 Connection Block

There is a row associated with each literal from $X \cup \overline{X}$, and three columns associated with each clause. In the connection block, which will not actually be traversed by the robot, there is one free position in each clause column, and



(a) Schematic View.



(b) Complete Example for the formula $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_1}) \wedge (x_3 \vee x_1 \vee x_2)$.

Fig. 9. The Push-* Construction.

all other positions are blocked. These free positions correspond to the literals forming the clause and are thus in the rows associated with these literals. See Figure 10 for an example. Because most positions on our grid will be blocked, we instead denote free positions, blockwise as polygons, so, e.g., in Figure 10 there are three free positions.

Proposition 9 *In the connection block there are n_i free positions in the row associated with x_i and $\overline{n_i}$ free positions in the row associated with $\overline{x_i}$ for $1 \leq i \leq k$.*

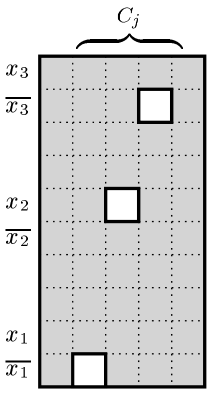


Fig. 10. Connections for $C_j = \overline{x_1} \vee x_2 \vee \overline{x_3}$.

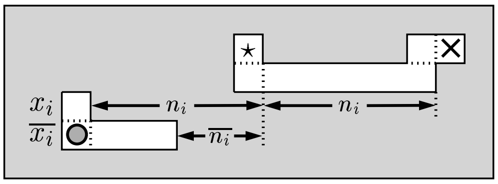


Fig. 11. Variable Gadget.

3.2 Variable Gadgets

See Figure 11. For each variable x_i there is one variable gadget in the variable block, representing both corresponding literals. The bottom two rows are associated with x_i and $\overline{x_i}$, respectively, as described in the previous section. The gadgets are placed diagonally on top of each other, from the bottom-left corner up to the top-right corner of the variable block, such that the goal position of the previous is directly below the start position of the next gadget. The gadgets are to be traversed one after another and accordingly the start and goal positions of each gadget (denoted by a small circle and cross, respectively) are also in the mentioned corners. The gadget's size depends on n_i: the height is 4 and the width is $2\,n_i + 2$.

Lemma 10

(1) The variable gadget for x_i can be traversed if and only if at least one of *row* x_i or *row* $\overline{x_i}$ *is pushed n_i or $\overline{n_i}$, respectively, times to the right.*
(2) The positions outside the gadgets cannot be changed, except for the $n_i + \overline{n_i}$ free positions in the connection block corresponding to x_i and $\overline{x_i}$.
(3) The only way to leave the gadget (its bounding box) is through the start and goal positions.

Proof. The "if" part of (1) is obvious. For the "only if" part, observe that initially the only pushing positions are in the x_i and (possibly) the $\overline{x_i}$ row where the robot can push right. This property does not change until the column of $\star$ is reached. Moreover, there are no free positions vertically outside the gadget, except for (possibly) below the start and above the goal position. Hence, (1) is true for the first gadget. For the remaining gadgets, we can argue analogously, as soon as we have proved (2). There are no free positions horizontally outside the gadget, except for the $n_i + \overline{n_i}$ free positions in the connection block to the right. This number is exactly the difference in coordinates between the rightmost free position in the free starting component and the right border of the gadget. Thus, the robot cannot pass the gadget's horizontal borders. The only remaining way to change positions outside the gadget is to push out some blocks from the start or goal position, which is easily seen to be impossible. Hence, (2) and (3) follow. □

Note that without the gap of width n_i between position $\star$ and the goal position, property (3) would not hold in general: the robot could push $\overline{n_i}$ times right in the bottommost row of the variable gadget, go up one row, and go right another n_i times.

3.3 Bridge Gadgets

There is only one bridge gadget, but it consists of three components placed within a large horizontal space. Therefore it has been divided into two blocks in the schematic view of Figure 9(a). Figure 12 shows the gadget as a whole, where the left part corresponds to the Up-Bridge and the two parts on the right form the Right-Bridge.

Let $h := 3\,\ell - 1$ denote the height of the clause block and $x := 10\,\ell - 2$ be the number of free positions in the clause block plus the h free positions in the lower Right-Bridge component. (To check the numbers, refer to Section 3.4. Recall that ℓ denotes the number of clauses.)

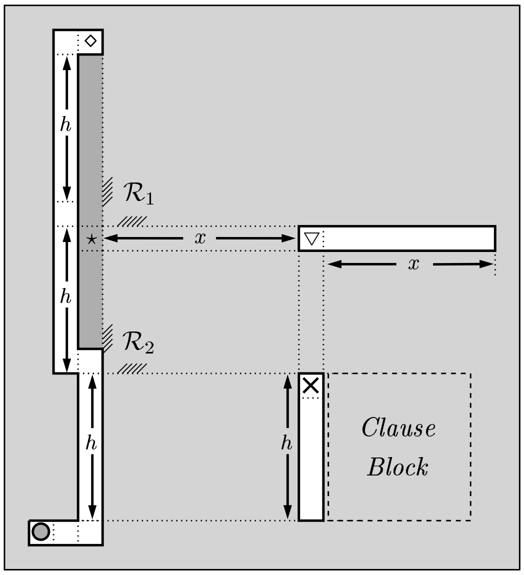


Fig. 12. Bridge Gadget.

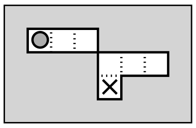


Fig. 13. Clause Gadget.

Lemma 11 *After traversing the bridge gadget, the following holds for any position p in the clause block: if there is a free position f in the same row or in the same column as p, then either f lies inside the clause block, or f is the goal position of the bridge block.*

Proof. There is a straightforward way for the robot to pass through the bridge gadget: move up the free positions to $\diamond$, then pushing down to $\star$, right to $\bigtriangledown$, and finally down to the goal position (cf. Figure 14). We will show in the following that this is basically the only way to traverse the gadget.

By the choice of x, there is no way to reach the goal column without using the free positions in the upper-right component (the one containing $\bigtriangledown$). But even in that row, we have to get one more free position between $\star$ and $\bigtriangledown$ in order to reach the right components. However, the region $\mathcal{R}_1$ that is above and right of the blocked part of that row between the bridge components is completely blocked. Hence, there is no way to push any of the blocks out there, except for the first one, at position $\star$. The robot must not push this block right, because then it would be impossible to remove the block from the row for the reason stated above. Hence, there is no way for the robot to enter the region $\mathcal{R}_2$ to the right of the left component and above the clause block, before the block at $\star$ has been removed from its row. Thus, there is no way to push away any of the shaded blocks below $\star$; and pushing them upwards does not help, because $\star$ would still be blocked. The only way to clear $\star$ is to push the shaded blocks down, completely blocking this column in the rows to the right of the clause block. Obviously, the robot has to go down h times from $\bigtriangledown$, blocking the positions to the right of the clause block in this column as well, except for the goal position. Also, the part above the clause block is blocked from going right to $\bigtriangledown$. The only remaining possibility to have free positions in some rows to the right of the clause block is when some blocks have been pushed into the clause block from the column of $\star$. But then at least one position in the column below $\bigtriangledown$ would be blocked and it would be impossible to reach the goal position. $\square$

3.4 Clause Gadgets

There is one clause gadget, as depicted in Figure 13, for every clause. They are aligned diagonally, from the top-left down to the bottom-right corner of the clause block, such that the goal position of the previous gadget is directly above the start position of the next gadget. In the last gadget, we save one free position by moving the goal position one row up. Thus, the total number of free positions in the clause block is $7\,\ell - 1$ and its height (in terms of rows) is $3\,\ell - 1$.

Lemma 12 *The robot can pass through a clause gadget if and only if it pushes down in one of the first three columns.*

Proof. The "if" part is obvious. For the "only if" part, consider the first (leftmost) clause gadget. By Lemma 11, there are no free positions to the left, except for the position immediately to the left of the start position. By construction, there are no free positions to the right, either. Hence, the claim is true for the first gadget. For the remaining gadgets, we can argue analogously that the same observation holds: even if all of the first three columns have a free position in the connection block (by construction there is at most one free position per column) and the robot uses all of them to push down, the bottommost position it reaches is the one diagonally left and below the goal position and hence immediately to the left of the starting position of the next gadget (if there is one). □

3.5 Putting Things Together

In the connection block, by Lemma 10 for any $i \in \{1, \ldots, k\}$ all free positions in at least one of the rows corresponding to x_i or $\overline{x_i}$ get blocked when the robot traverses the variable block. On the other hand, by Lemma 12 the gadget of a clause can be traversed only if for one of the three literals forming the clause there is a free position in the connection block. Thus, if the robot found its way through the puzzle, we can construct a satisfying assignment for the 3-Sat formula by setting those literals to true that had some free positions in the connection block after the robot traversed the variable block. It might happen that for some i all free positions in the connection block corresponding to both x_i and $\overline{x_i}$ are blocked, but then we can set x_i arbitrarily.

On the other hand, if we have a satisfying assignment, by Lemma 10 the robot can pass through the variable block by pushing only those rows associated with literals which have been set to false. Then the robot will find its path to the goal position as described in Lemmata 11 and 12 (cf. Figure 14). Altogether, this yields the following lemma.

Lemma 13 *The constructed* Push-X *puzzle is solvable if and only if the 3-*Sat *formula has a satisfying assignment.*

Now NP-hardness of Push-X follows immediately, noting that its size is polynomial, in fact at most quadratic, in the number of variables and clauses of the 3-Sat formula.

Theorem 14 Push-* *is NP-hard.*

Corollary 15 PUSH-*-X, PUSHPUSH-*, *and* PUSHPUSH-*-X *are NP-hard.*

Proof. As in Corollary 8.

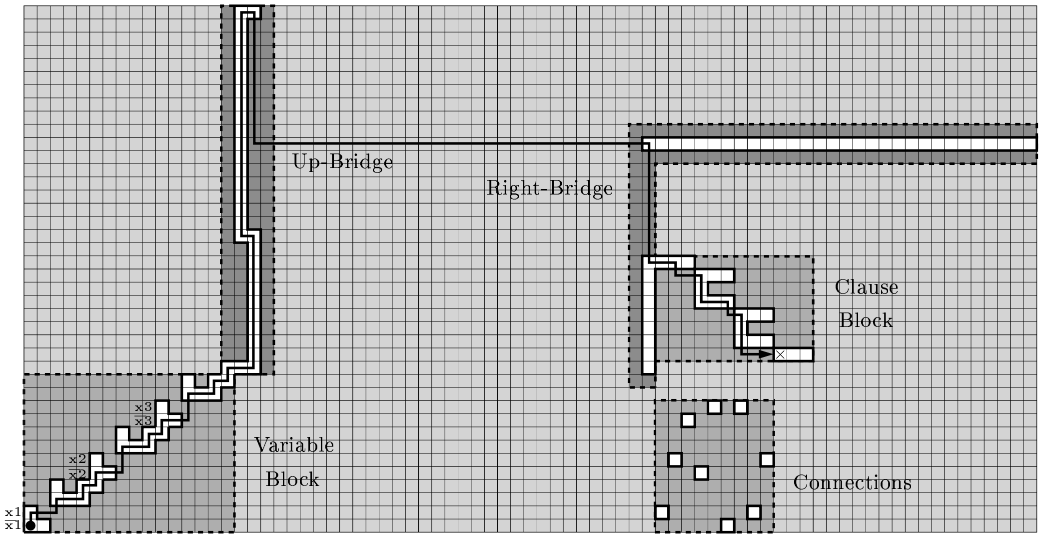


Fig. 14. A solution path for the example from Figure 9(b).

4 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-* [3].

Another still-open question is whether there is an "interesting" pushing-block puzzle that is still tractable. Up to now, the only such problem known to be in P is if the robot is restricted to monotonic paths [9], e.g., it can only move up and right.

References

[1] BATTISTA, G. D., EADES, P., TAMASSIA, R., AND TOLLIS, I. G. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] Demaine, E. D., and Hoffmann, M. Pushing blocks is NP-complete for noncrossing solution paths. In *Proc. 13th Canad. Conf. Comput. Geom.* (2001), pp. 65–68.

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

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

[10] Dor, D., and Zwick, U. Sokoban and other motion planning problems. *Comput. Geom. Theory Appl. 13*, 4 (1999), 215–228.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

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