PushPush is NP-hard in 2D

Erik D. Demaine Martin L. Demaine* Joseph O'Rourke†

January 24, 2000

Abstract

We prove that a particular pushing-blocks puzzle is intractable in 2D, improving an earlier result that established intractability in 3D [OS99]. The puzzle, inspired by the game *PushPush*, consists of unit square blocks on an integer lattice. An agent may push blocks (but never pull them) in attempting to move between given start and goal positions. In the PushPush version, the agent can only push one block at a time, and moreover, each block, when pushed, slides the maximal extent of its free range. We prove this version is NP-hard in 2D by reduction from SAT.

1 Introduction

There are a variety of "sliding blocks" puzzles whose time complexity has been analyzed. One class, typified by the 15-puzzle so heavily studied in AI, permits an outside agent to move the blocks. Another class falls more under the guise of motion planning. Here a robot or internal agent plans a path in the presence of movable obstacles. This line was initiated by a paper of Wilfong [Wil91], who proved NP-hardness of a particular version in which the robot could pull as well as push the obstacles, which were not restricted to be squares. Subsequent work sharpened the class of problems by weakening the robot to only push, never pull obstacles, and by restricting all obstacles to be unit squares. Even this version is NP-hard when some blocks may be fixed to the board (made unpushable) [DO92].

One theme in this research has been to establish stronger degrees of intractability, in particular, to distinguish between NP-hardness and PSPACE-completeness, the latter being the stronger claim. The NP-hardness proved in [DO92] was strengthened to PSPACE-completeness in an unfinished manuscript [BOS94]. More firm are the results on Sokoban, a computer game that restricts the pushing robot to only push one block at a time, and requires the

*Dept. Comput Sci., Univ. Waterloo, Waterloo, Ontario N2L 3G1, Canada. {eddemaine, mldemaine}@uwaterloo.ca.

†Dept. Comput. Sci., Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu. Supported by NSF grant CCR-9731804.

storing of (some or all) blocks into designated "storage locations." This game was proved NP-hard in [DZ95], and PSPACE-complete by Culberson [Cul98].

Here we emphasize another theme: finding a nontrivial version of the game that is *not* intractable. To date only the most uninteresting versions are known to be solvable in polynomial time, for example, where the robot's path must be monotonic [DO92]. We explore a different version, again inspired by a computer game, PushPush.[1] The key difference is that when a block is pushed, it necessarily slides the full extent of the available empty space in the direction in which it was shoved. This further weakens the robot's control, and the resulting puzzle has certain polynomial characteristics. It was established in [OS99] that the problem is intractable in 3D, but its status in 2D was left open in that paper. Here we settle the issue by extending the reduction to 2D.

2 Problem Classification

The variety of pushing-block puzzles may be classified by several characteristics:

1. Can the robot pull as well as push?
2. Are all blocks unit squares, or may they have different shapes?
3. Are all blocks movable, or are some fixed to the board?
4. Can the robot push more than one block at a time?
5. Is the goal for the robot to move from s to t, or is the goal for the robot to push blocks into storage locations?
6. Do blocks move the minimal amount, exactly how far they are pushed, or do they slide the maximal amount of their free range?
7. The dimension of the puzzle: 2D or 3D?

If our goal is to find the weakest robot and most unconstrained puzzle conditions that still lead to intractability, it is reasonable to consider robots who can only push (1), and to restrict all blocks to be unit squares (2), as in [DO92, DZ95, Cul98], for permitting robots to pull, and permitting blocks of other shapes, makes it relatively easy to construct intractable puzzles. It also makes sense to explore the goal of simply finding a path (5) as in [Wil91, DO92], rather than the more challenging task of storing the blocks as in Sokoban [DZ95, Cul98].

Restricting attention to these choices still leaves a variety of possible problem definitions. If the robot can only move one block at a time, then the distinction between all blocks movable and some fixed essentially disappears, because 2×2 clusters of blocks are effectively fixed to a robot who can only push one. If all

[1] The earliest reference we can find to the game is a version written for the Macintosh by Alan Rogers and C.M. Mead III, Copyright 1994, `http://www.kidsdomain.com/down/mac/pushpush.html`. Another version for the Amiga was written by Luigi Recanatese in 1997, `http://de.aminet.net/aminet/dirs/game_think.html`.

blocks are movable and the robot can push more than one at a time, then the blocks should be confined to a rectangular frame.

The version explored in this paper superficially seems that it might lend itself to a polynomial-time algorithm: the robot can only push one block (4), all blocks are pushable (3), and finally, the robot's control over the pushing is further weakened by condition (6): once pushed, a block slides (as without friction) the maximal extent of its free range in that direction. Allowing the robot to move in 3D gives it more "power" than it has in 2D, so the natural question after [OS99] is to explore the weaker option of condition (7): PushPush in 2D.

Because our proof is a direct extension of the proof for 3D in [OS99], we repeat through Section 6 the 3D construction in that paper (with a minor simplification), and in Section 7 show a similar reduction holds in 2D as well. (Both reductions are from SAT, i.e., satisfiability of formulas in conjunctive normal form.) A summary of related results is presented in the final section.

3 Elementary Gadgets

First we observe, as mentioned above, that any 2×2 cluster of movable blocks is forever frozen to a PushPush robot, for there is no way to chip away at this unit. This makes it easy to construct "corridors" surrounded by fixed regions to guide the robot's activities. We will only use corridors of width 1 unit, with orthogonal junctions of degree two, three, or four. We can then view a particular PushPush puzzle as an orthogonal graph, whose edges represent the corridors, understood to be surrounded by sufficiently many 2×2 clusters to render any movement outside the graph impossible. We will represent movable blocks in the corridors or at corridor junctions as circles.

We start with three elementary gadgets.

3.1 One-Way Gadget

A *One-Way* gadget is shown in Fig. 1a. It has these obvious properties:

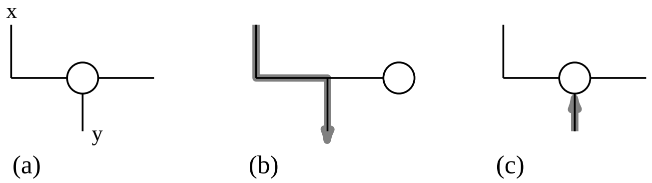

Figure 1: One-Way gadget: permits passage from x to y but not from y to x.

Lemma 1 *In a One-Way gadget, the robot may travel from point x to point y, but not from y to x. (After travelling from x to y, however, the robot may subsequently return from y to x.)*

Proof: The block at the degree-three junction may be pushed into the storage corridor when approaching from x, as illustrated in Fig. 1b, but the block may not be budged when approaching from y (Fig. 1c). □

3.2 Fork Gadget

The *Fork gadget*[2] shown in Fig. 2a presents the robot with a binary choice, the proverbial fork in the road:

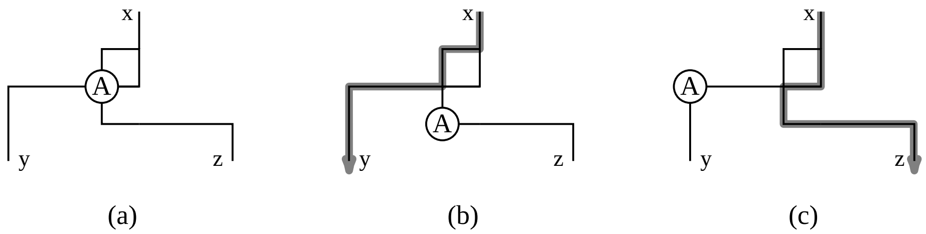

Figure 2: Fork gadget: Robot may pass (b) from x to y or (c) from x to z, but each seals off the other possibility.

Lemma 2 *In a Fork gadget, the robot may travel from point x to y, or from x to z, but if it chooses the former it cannot later move from y to z, and if it chooses the latter it cannot later move from z to y. (In either case, the robot may reverse its original path.)*
Proof: Fig. 2b shows the only way for the robot to pass from x to y. Now the corridor to z is permanently sealed off by block A. Fig. 2c shows the only way to move from x to z, which similarly seals off the path to y. □

Note that in both these gadgets, the robot may reverse its path, a point to which we will return in Section 7.

3.3 3D Crossover Gadget

Crossovers are trivial in 3D, as shown in Fig. 3.

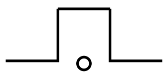

Figure 3: 3D crossover. The central small circle is a wire orthogonal to the plane of the figure.

[2] This is a simplification of the functionally equivalent gadget used in [OS99].

4 Variable-Setting Component

The robot first travels through a series of variable-setting components, each of which follows the structure shown in Fig. 4: a Fork gadget, followed by two paths, labeled T and F, each with attached *wires* exiting to the right, followed by a re-merging of the T and F paths via One-Way gadgets. 3D crossovers are illustrated in this and subsequent figures by broken-wire underpasses.

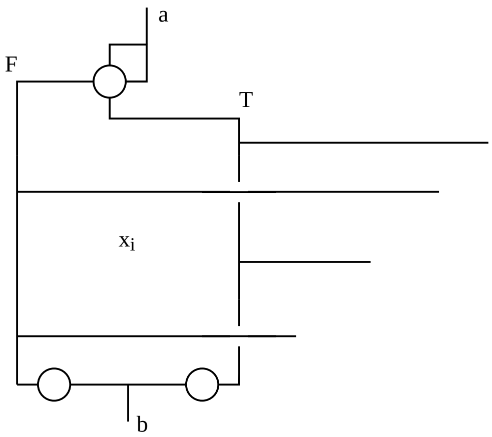

Figure 4: (a) Variable x_i component.

Lemma 3 *The robot may travel from a to b only by choosing either the* T*-path, or the* F*-path, but not both. Whichever* T/F*-path is chosen allows the robot to travel down any wires attached to that path, but down none of the wires attached to the other path.*
Proof: The claims follow directly from Lemma 2 and Lemma 1. □

5 Clause Component

The clause component shown in Fig. 5a cannot be traversed unless one or more blocks ("keys") are pushed in from the left along the attached horizontal wires.

Lemma 4 *The robot may only pass from x to y of a clause component if at least one block is pushed into it along an attached wire (a, b, or c in Fig. 5a).*
Proof: Block A is necessarily pushed by the robot starting at x. This block will clog exit at y (Fig. 5b) unless its sliding is stopped by a block pushed in on an attached wire. □

The basic mechanism that gives a clause component its functionality will be reused in several guises in Section 7, so we pause to redescribe it. Essentially

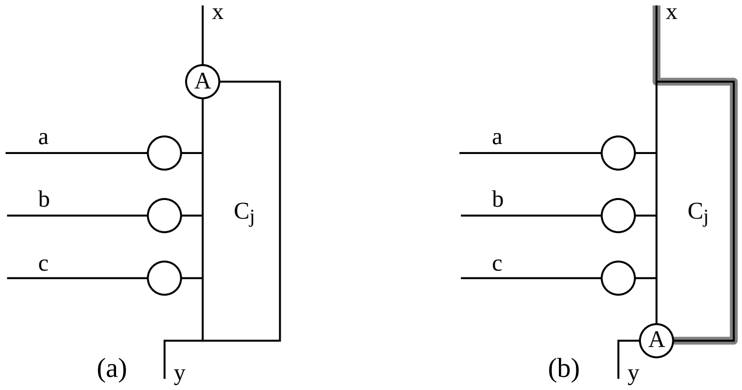

Figure 5: (a) Clause C_j component. (b) Passage thwarted without an "inserted" key block.

the component is a *lock* with three *keys*, which we identify with the three blocks on the a, b, and c wires. A necessarily-pushed block A is characteristic of locks, as is an alternate path around the spot(s) where the key(s) come to rest.

6 Complete SAT Reduction

The complete construction for four clauses $C_1 \wedge C_2 \wedge C_3 \wedge C_4$ is shown in Fig. 6. Two versions of the clauses are shown in the figure: an unsatisfiable formula (the dark lines), and a satisfiable formula (including the shaded x_2 wire):

$$(x_1 \vee x_2) \wedge (x_1 \vee \sim x_2) \wedge (\sim x_1 \vee x_3) \wedge (\sim x_1 \vee \sim x_3) \tag{1}$$

$$(x_1 \vee x_2) \wedge (x_1 \vee \sim x_2) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (\sim x_1 \vee \sim x_3) \tag{2}$$

Here we are using $\sim x$ to represent the negation of the variable x.

A path from s to t in the satisfiable version is illustrated in Fig. 7.

Theorem 1 *PushPush is NP-hard in 3D.*
Proof: The construction clearly ensures, via Lemmas 3 and 4, that if the simulated Boolean expression is satisfiable, there is a path from s to t, as illustrated in Fig. 7. For the other direction, suppose the expression is unsatisfiable. Then the robot can reach t only by somehow "shortcutting" the design. The design of the variable components ensures that only one of the T/F paths may be accessed. The crossovers ensure there is no "leakage" between wires. The only possible thwarting of the design would occur if the robot could travel from a clause component back to set a variable to the opposite Boolean value. But each variable-clause wire contains a block that prevents any such leakage. □

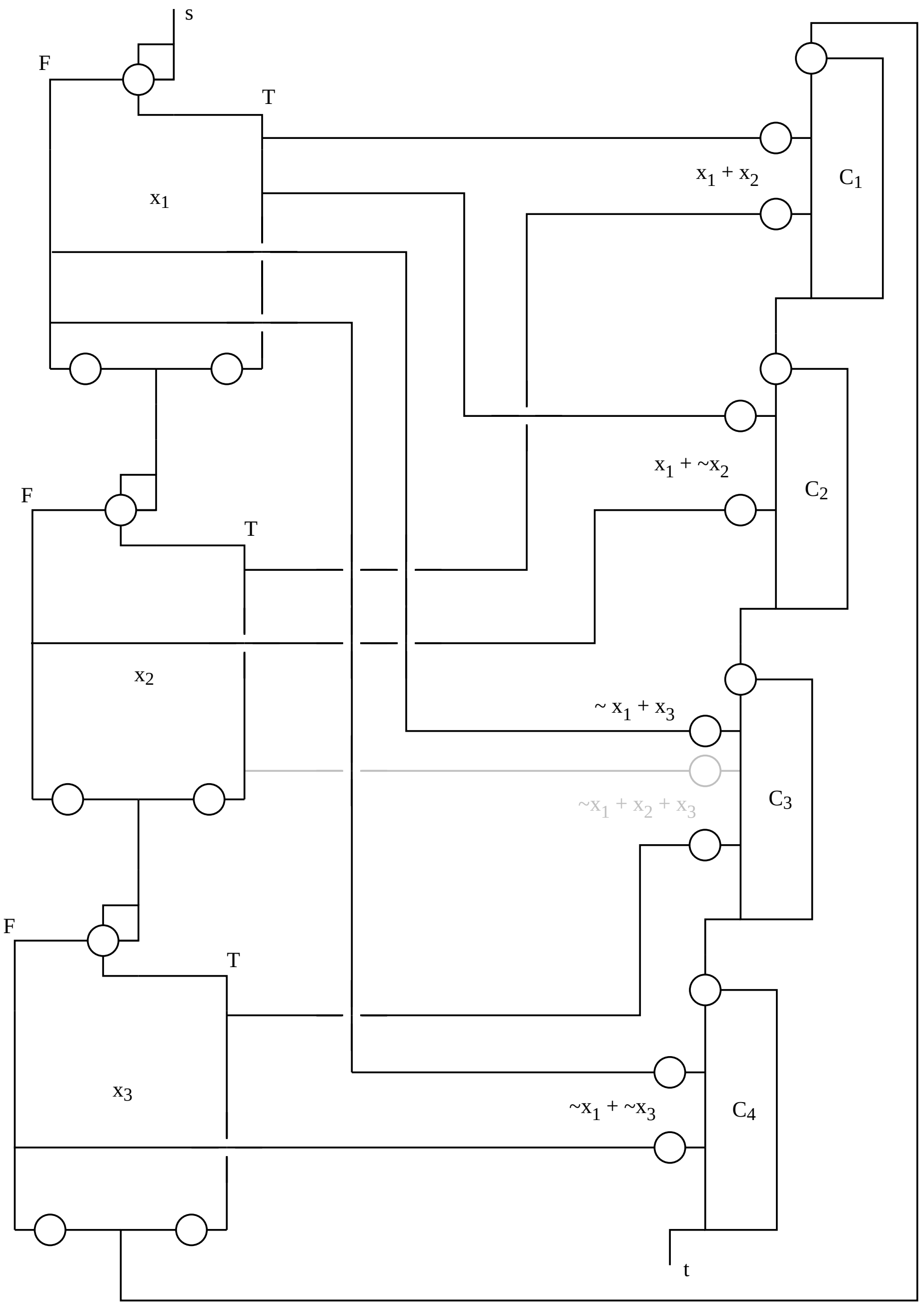

Figure 6: Complete construction for the formulas in Eq. (1) and Eq. (2) (including the shaded portion).

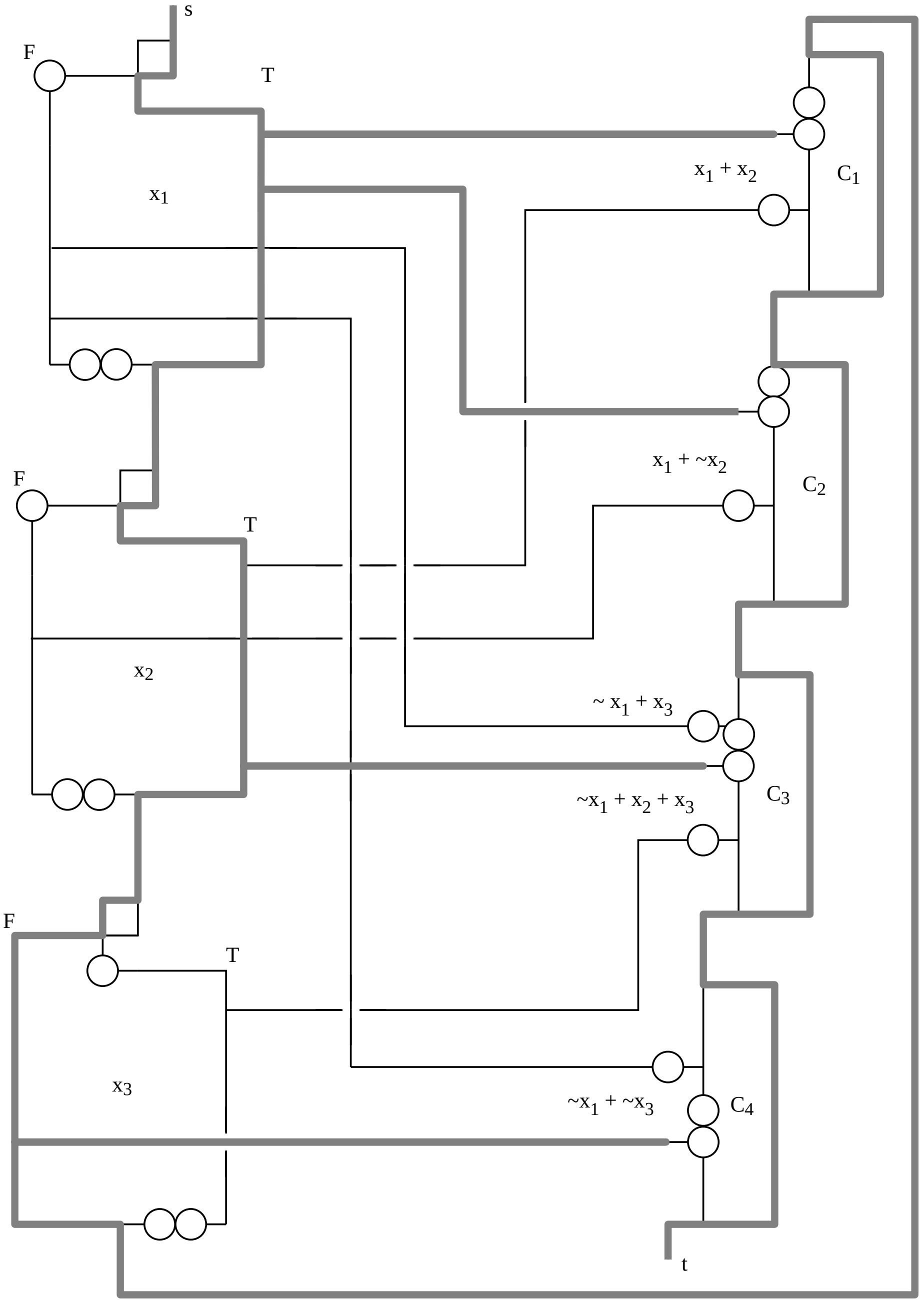

Figure 7: Solution path for Eq. (2).

7 2D Crossovers

We now modify the 3D SAT reduction to a 2D SAT reduction by replacing the 3D crossovers with appropriate 2D crossovers; it is only here that we deviate substantively from [OS99]. Note that there are two distinct types of crossovers used in the construction:

1. *FT-crossovers*: A horizontal wire from an F-wire in a variable unit crosses the vertical T-wire of the same variable unit.
2. *VC-crossovers*: A horizontal wire from some variable unit to a clause unit crosses a vertical wire from some other variable unit to some clause unit.

The FT-crossovers are significantly different from the VC-crossovers in that the former are traversed in one direction or the other but never both. This is because once the robot chooses the T-wire, it can never get into the F-wire, and vice versa (Lemma 3). Thus a limited crossover suffices here.

There is another approach to handling the FT-crossovers: reduction from "Planar 3-SAT" [Lic82] (as used, e.g., in Dor and Zwick's NP-hardness proof [DZ95]) permits eliminating this type of crossover entirely. However, we do not pursue that tack, for use of Planar 3-SAT introduces additional crossovers in the final clause-threading path. Instead we develop a limited crossover gadget for FT-crossovers, which may serve as an introduction to the more demanding VC-crossovers in Section 7.2.

7.1 XOR Crossover

We call the limited crossover gadget an *XOR Crossover*; it is shown in Fig. 8(a).

Lemma 5 *An XOR Crossover gadget permits either horizontal, rightward passage without leakage into the vertical channel, or vertical, downward passage without leakage into the horizontal channel, but not both. In either case, the passage may afterwards be traversed an arbitrary number of times in the same direction.*

Proof: Entering the unit from the F-wire to the left pushes block A into the vertical channel (Fig. 8(b)), which, together with block B, disallows entry into the vertical T-wire ((c) of the figure). Similarly, entering the unit from the T-wire at the top pushes block B to clog the horizontal channel (d), preventing leakage into either the F-wire or the clause unit (e). □

7.2 Locking Door Unit

VC-crossovers do not have the exclusive-or property that makes FT-crossovers so easy to handle. They may need to be traversed in either direction. However, note that VC-crossovers need only be traversed once in each of the four directions: from a variable component x_i, to deposit a key into a clause component

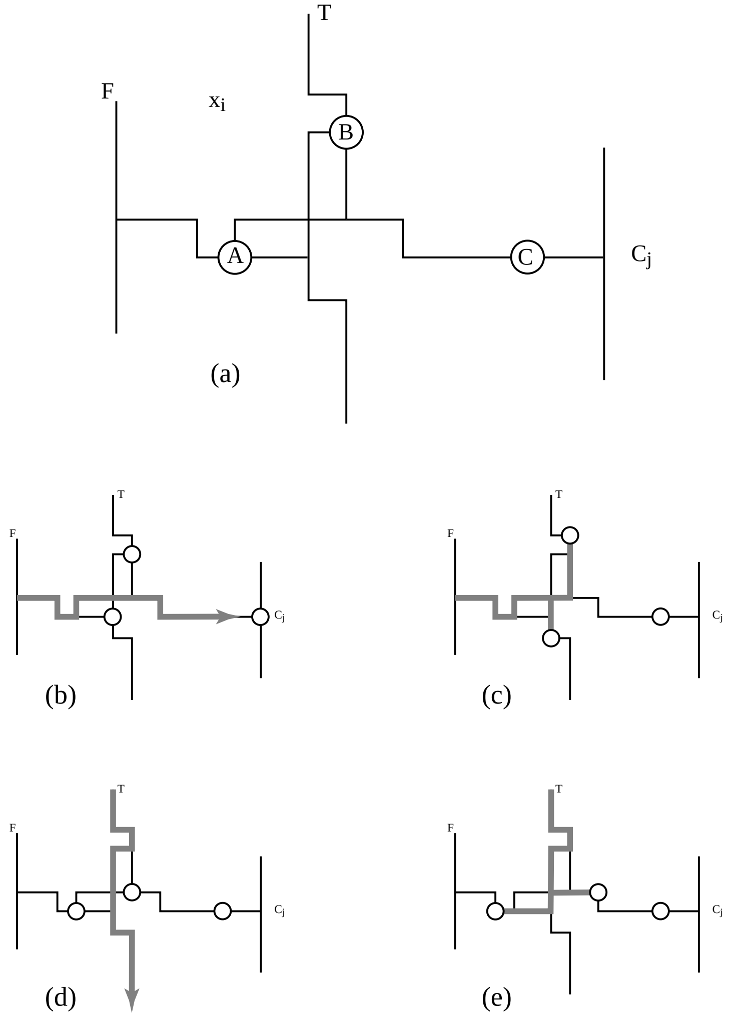

Figure 8: (a) XOR Crossover gadget. (b,c): Horizontal passage; (d,e): Vertical passage.

C_j, and returning back to x_i; and later from x_k, $k \neq i$, to some C_l, and back to x_k. The design of such a crossover is the most complex part of our construction, and will proceed in several stages.

The most important gadget used to build this crossover is one that permits passage in one direction, but then prevents return in the other direction. We call this a *Locking Door Unit*; it is shown in Fig. 9(a). Like a One-Way gadget, it may only be traversed in one direction. But unlike that gadget, once traversed it becomes a permanently locked door with respect to both directions. A more accurate name for the gadget might be "unidirectional, single-use, self-closing and self-locking door." Note that the unit contains a lock mechanism (similar to that used in a clause component) centered around point b, which requires the key block B to permit passage.

Lemma 6 *Upon first encounter, the robot may pass from x to y through a locking door unit, and not y to x; but once through, the unit becomes impassable in either direction.*

Proof: We first argue that the unit may be passed through as claimed. The robot first moves from x to below block A, which it pushes upward to point a. This seals off return passage. It then pushes block B down to point b, as shown in Fig. 9(b). B serves as a key to an exit door. It then "unlocks" that door by pushing block C to abut against B. It may then travel around CB, push block D to the right, and reach the exit y. See (c) of the figure. Note that it now impossible for the robot to return from y to x; moreover it is impossible for the robot to retraverse the unit from x to y again, in both cases because of block A at point a.

Next we argue that the above is the only way to traverse the unit. It may not be traversed from y to x because of the One-Way gadget represented by block D. We consider the four options of what to do with block A after entering at x:

1. Push A upward. This leads to the solution just described.

2. Do not move A. Then B is unreachable, and the only option is to push block C rightwards. But without the key B at b, C slides over to abut D and clog the exit. See (d) of the figure.

3. Push A downward. Then A slides down to touch C. Now the robot can only push the key B down to b ((e) of the figure), but then the door cannot be "unlocked" with C as that is now inaccessible.

4. Push A rightward. Then A hits B, and prevents the use of the key at b. Now, similar to (d) of the figure, C slides all the way to D and prevents exit.

This exhausts the options and establishes the claims of the lemma. $\square$

We will use the symbols illustrated in Fig. 10(a-b) to represent a locking door before (a) and after (b) passage.

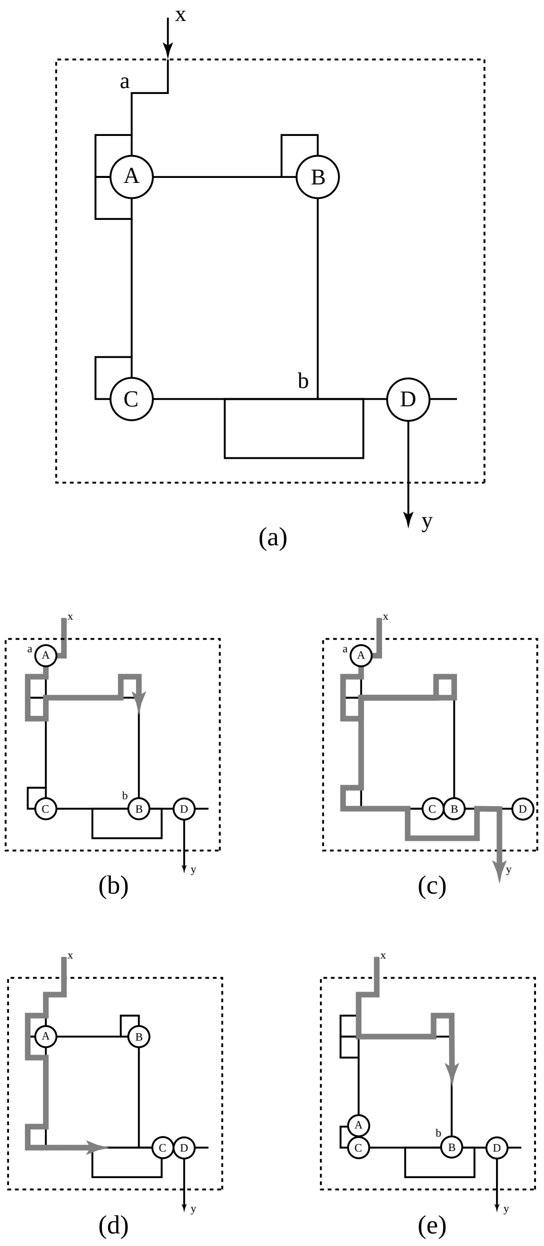

Figure 9: (a) Locking door unit. (b,c): Passage through unit; (d,e): Bad attempts.

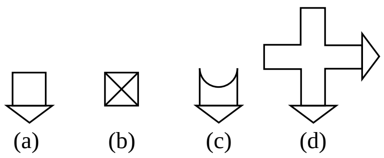

Figure 10: Symbols: (a) Locking door prior to passage. (b) After passage. (c) Double lock unit prior to passage. (d) Unidirectional crossover unit.

7.3 Double Lock Unit

We next detail a gadget that behaves like a locking door, in that passage is permitted once in one direction, but which requires an external key to operate. We call this a *Double Lock Unit*. It is illustrated in Fig. 11(a).

Lemma 7 *The robot may pass from x to y through a double lock unit only if a block A (the external key) is first pushed down the entrance x-wire. A double lock may not be traversed from y to x, and once traversed forwards from x to y, the unit becomes impassable in either direction.*

Proof: We call the two locks in the unit the A- and B-locks. The former consists of the key block A, block C, and the wires near point a; the latter consists of the key block B, block D, and the wires near point b. Passage through the unit when there is an external block A is shown in Fig. 11(b). Key block A is pushed to point a, and then the robot goes through the locking door unit to reach B. This B-key is pushed to point b. Now both locks have keys. The robot then pushes block C leftwards and passes through the A-lock, pushes block D rightwards and passes through the B-lock, pushes block E out of the way, and finally exits at y.

Suppose there is no external A block, and the robot attempts to traverse the unit. To reach y, the robot must pass through the B-lock. If it approaches from the left without first pushing down the B-key, then block D will be pushed rightwards and clog exit at point d. So the key block B must be pushed down to point b. The only way to reach B is via the locking door above it (because block C prevents access from the left). But passing through the locking door leaves the robot no alternative but to push C leftwards, as in in Fig. 11(c). This blocks access to the B-lock.

The initial position of block E ensures that the unit cannot be traversed from y to x upon first encounter. That it cannot be traversed in either direction after forward passage is clear by inspection of Fig. 11(b). □

We will use the symbol shown in Fig. 10(c) to represent a double lock unit prior to passage,[3] and again the 'X' in (b) to represent the unit after passage.

[3] The circular "bite" taken out of the arrow is to remind us that passage requires an external key.

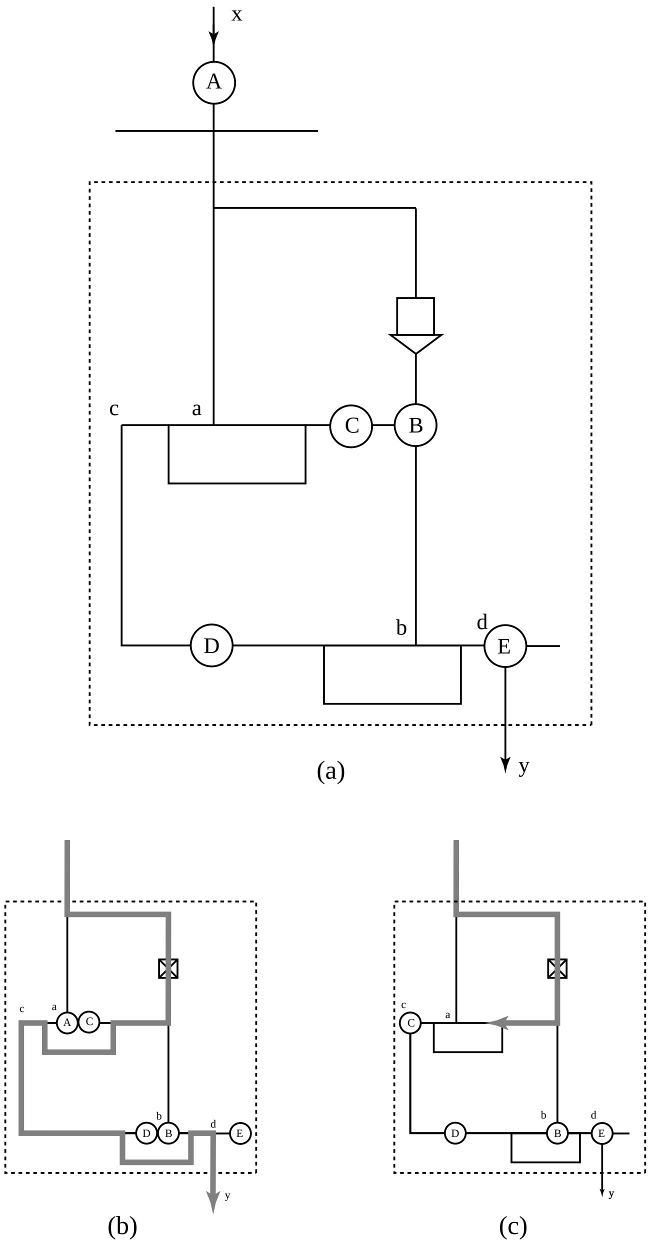

Figure 11: (a) Double lock unit. (b) Passage through with A-key. (c) Attempt without A-key.

7.4 Unidirectional Crossover Gadget

We are finally prepared to design a gadget that removes the exclusive-or limitation of the XOR crossover gadget. We call this a *Unidirectional Crossover gadget*; see Fig. 12(a). Call the directed horizontal path (x_1, x_2) the *forward x-path*, and the directed vertical path (y_1, y_2) the *forward y-path*. The *backward* paths are the reverse of these. The point where the paths meet we call the *junction* of the gadget. The forward x-path consists of a locking door followed by a key block X prior to the junction and a double lock after the junction; the y-path is structured similarly.

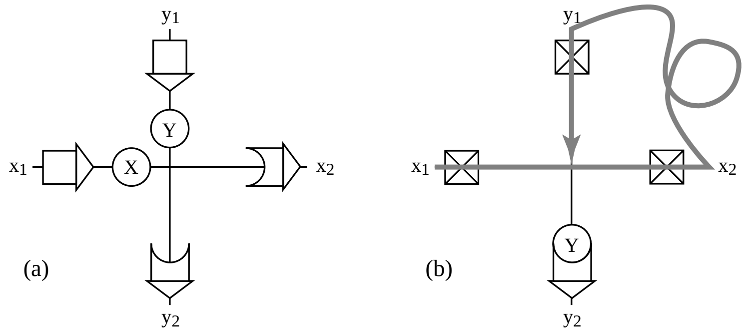

Figure 12: (a) Unidirectional Crossover. (b) No leakage.

We denote the unidirectional crossover gadget by the symbol in Fig. 10(d).

Lemma 8 *A unidirectional crossover gadget may be traversed along the forward x-path (but not the backward x-path), and along the forward y-path (but not the backward y-path), in either order. Once either path is traversed, that path (but not the other) becomes impassable in either direction. If the junction is approached from x_1, then the robot may not from there reach either y_1 or y_2 without first passing through the x-path to x_2; and symmetrical claims hold after approaching the junction from y_1.*

Proof: The design is symmetrical, so we need only establish its properties for the x-path. If the robot is at x_1, it can only enter the unit by passing through the locking door, which then is permanently sealed behind it. The robot must then push the X block into the double lock unit on the x-path. X serves as the external key for that unit, and permits passage as in Lemma 7. Thus passage along the forward x-path is possible. The reverse passage is not possible, neither initially (because a double lock cannot be traversed backwards), nor after forward passage (because the double lock then becomes closed).

Unlike the XOR crossover, passage along the x-path does not affect the ability to later traverse the forward y-path. Finally, note that when the robot reaches the junction starting from x_1, it may not "leak" into the y-path: its upward movement is stopped by the locking door, and its downward movement is prevented by the inability to pass through the double lock without the Y-key—the robot is on the "wrong side" of the key. Similarly, if the robot first traverses the x-path, and then reaches the junction via the y-path, then it may not "leak" into the x-path, as Fig. 12(b) illustrates. □

7.5 Bidirectional Crossover Gadget

We can finally construct a VC-crossover, which we call a *Bidirectional Crossover Gadget*; see Fig. 13. Again call the directed horizontal path (x_1, x_2) the *forward*

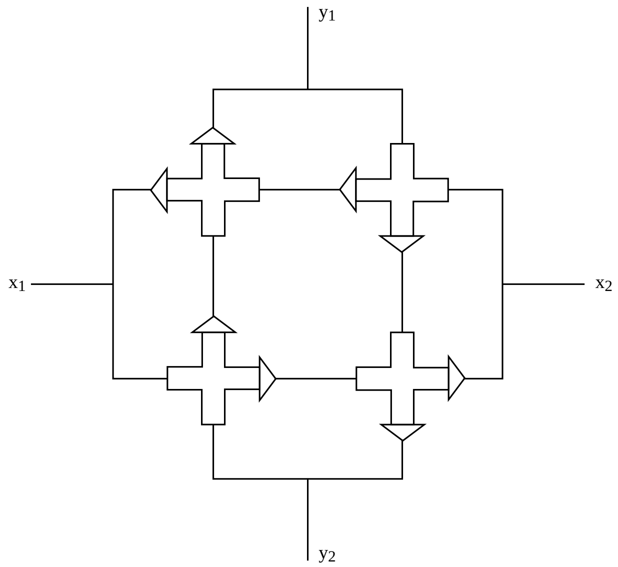

Figure 13: Bidirectional crossover.

x-path, and similarly for the *forward y-path* and the *backward* paths.

Lemma 9 *A bidirectional crossover may be traversed as many as four times with any combination, in any order, of the forward x-path, the backwards x-path, the forward y-path, and the backwards y-path. Furthermore, no leakage between an x-path and y-path is possible.*

Proof: The claim follows immediately from the construction and the properties of the unidirectional crossover gadget established in Lemma 8. For example, the forward x-path is traversed by passing through the bottom two unidirectional gadgets. Later the reverse y-path can be traversed by passing through the left two unidirectional gadgets. This still leaves it possible to later traverse the forward y-path and the reverse x-path. □

7.6 2D Construction

Replacing each FT-crossover by an XOR crossover gadget (Fig. 8), and each VC-crossover by a bidirectional crossover gadget (Fig. 13) completes the 2D construction. (Note we could replace the FT-crossovers by bidirectional crossover units, although the extra complexity of this gadget is not needed here.) The entire construction may be traversed from s to t iff the represented Boolean formula is satisfiable. We have proved:

Theorem 2 *PushPush is NP-hard in 2D.*

We leave it an open question whether this theorem can be strengthened in either direction: either by proving PushPush is in NP, in which case it is NP-complete, or by showning that PushPush is PSPACE-complete.

8 Summary

We conclude by summarizing in Table 1 previous work according to the classification scheme offered in Section 2. The first four lines show previous results. The next two are the results from [OS99]. (The 2D storage result is, incidentally, not difficult.) The boldface line of the table is the result of this paper.

The last two lines list open problems. The penultimate line reposes the open question from [DO92]: Is the problem where all blocks are movable and the robot can push k blocks, sliding the minimal amount, intractable in 2D? The last line presents a new open problem, which we call *Push-1*: Is the problem where all blocks are movable, the robot can only push 1 block at time, pushing it the minimal amount, intractable in 2D? This differs from the problem addressed in this paper only in altering the sliding from *max* to *min*. Both of these open problems remain candidates for being solvable in polynomial time, the former because it seems difficult to construct gadgets when the robot can destroy them, the latter because without sliding the maximal extent, it seems difficult to design gadgets with the requisite functionality.

1 *Push?*	2 *Blocks*	3 *Fixed?*	4 #	5 *Path?*	6 *Sliding*	7 *Dim*	*Complexity*
pull	L	fixed	k	path	min	2D	NP-hard [Wil91]
push	unit	fixed	k	path	min	2D	NP-hard [DO92]
push	unit	movable	1	storage	min	2D	NP-hard [DZ95]
push	unit	movable	1	storage	min	2D	PSPACE [Cul98]
push	unit	movable	1	path	*max*	3D	NP-hard [OS99]
push	unit	movable	1	storage	*max*	2D	NP-hard [OS99]
push	**unit**	**movable**	**1**	**path**	***max***	**2D**	**NP-hard**
push	unit	movable	k	path	min	2D	open [DO92]
push	unit	movable	1	path	min	2D	open

Table 1: Pushing block problems.

Acknowledgements. We thank Therese Biedl for helpful discussions. The third author acknowledges many insights from meetings of the Smith Problem Solving Group.[4]

References

[BOS94] D. Bremner, J. O'Rourke, and T. Shermer. Motion planning amidst movable square blocks is PSPACE complete. Draft, June, 1994.

[Cul98] J. Culberson. Sokoban is PSPACE-complete. In *Proc. Internat. Conf. Fun with Algorithms*, pages 65–76, Elba, Italy, June 1998. Carleton Scientific.

[DO92] A. Dhagat and J. O'Rourke. Motion planning amidst movable square blocks. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 188–191, 1992.

[DZ95] D. Dor and U. Zwick. Sokaban and other motion planning problems. `http://www.math.tau.ac.il/~ddorit/`, 1995.

[Lic82] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[OS99] J. O'Rourke and the Smith Problem Solving Group. PushPush is NP-hard in 3D. Technical Report 064, Smith College, Northampton, MA, November 1999. Paper cs.CG/9911013, arXiv.org e-print archive, `http://arXiv.org/abs/cs.CG/9911013`.

[Wil91] G. Wilfong. Motion planning in the presence of movable obstacles. *Ann. Math. Artif. Intell.*, 3:131–150, 1991.

[4] Beenish Chaudry, Sorina Chircu, Elizabeth Churchill, Alexandra Fedorova, Judy Franklin, Biliana Kaneva, Haley Miller, Anton Okmianski, Irena Pashchenko, Ileana Streinu, Geetika Tewari, Dominique Thiébaut, Elif Tosun.