# PushPush is NP-hard in 3D

Joseph O'Rourke[*]
and The Smith Problem Solving Group[†]

November 28, 1999

**Abstract**

We prove that a particular pushing-blocks puzzle is intractable in 3D. 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 3D by reduction from SAT. The corresponding problem in 2D remains open.

## 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 [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 a 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. of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.-smith.edu. Research supported by NSF Grant CCR-9731804.

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

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. 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. We prove it is intractable in 3D, but leave the question of whether it is polynomial in 2D an open problem.

## 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 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 disappears, because 2x2 clusters of blocks are effectively fixed to a robot who can only push one. If all 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. We show the problem is intractable in 3D, and discuss the 2D version in the final section.

# 3 Elementary Gadgets

First we observe, as mentioned above, that any 2x2 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 2x2 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:

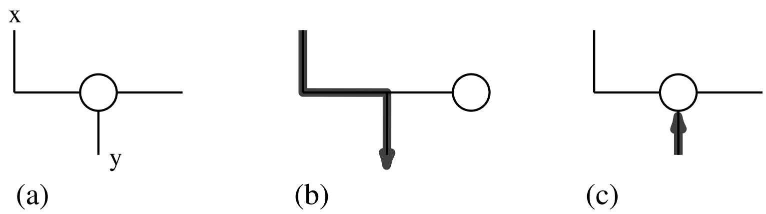


Figure 1: One-Way gadget: permits passage from $x$ to $y$ but not from $y$ to $x$.

**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 shown in Fig. 2a presents the robot with a binary choice, the proverbial fork in the road:

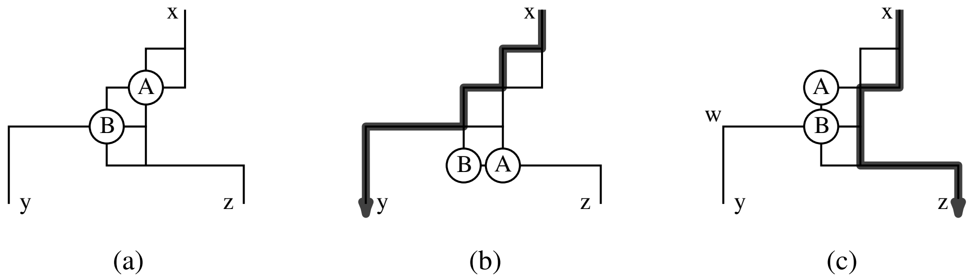


Figure 2: Fork gadget: Robot may pass (b) from $x$ to $y$ or (c) from $x$ to $z$, but each seals off the other possibility.

**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. Fig. 2c shows the only way to move from $x$ to $z$. Here any attempt later to access the corridor leading to $y$ will necessarily push block $B$ to corner $w$, sealing off $y$. $\square$

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.

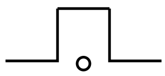

Figure 3: 3D crossover. The central small circle is a wire orthogonal to the plane of the figure.

## 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 the T and F paths via One-Way gadgets. 3D crossovers are illustrated in this and subsequent figures by broken-wire underpasses.

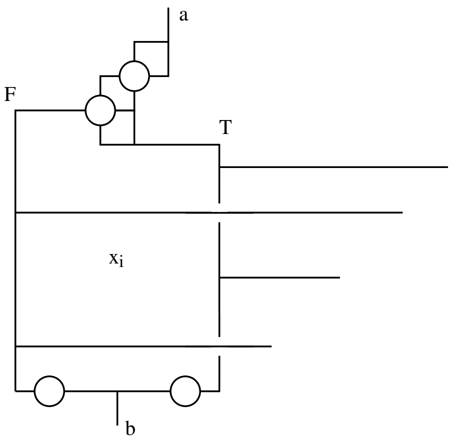


Figure 4: (a) Variable $x_i$ component.

**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 are pushed in from the left along the attached horizontal wires.

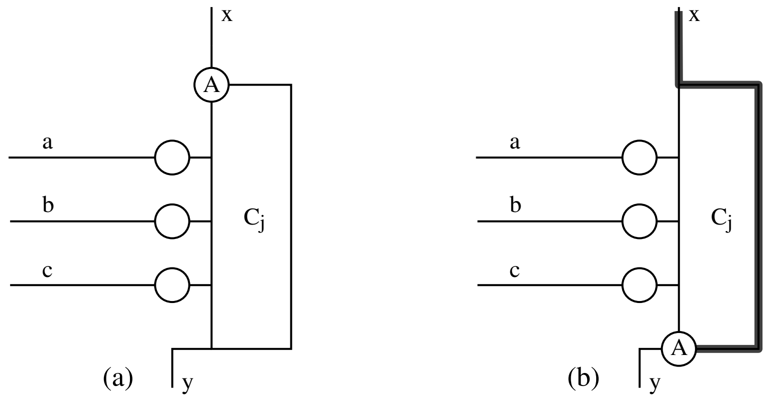


Figure 5: (a) Clause $C_j$ component.

**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. □

## 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. □

## 7 PushPush in 2D

It is an intriguing question whether the 2D version of this problem is intractable. One result in this direction is easy to obtain:

**Theorem 2** *The storage version of PushPush is NP-hard in 2D.*
**Proof:** In the storage version of PushPush,[1] the robot must fill certain storage locations with blocks, as in Sokoban. It is then easy to obtain an NP-hardness proof along the lines of the NP-hardness proof of Dor and Zwick [DZ95]. Rather than reducing from SAT, reduce from "Planar 3-SAT" [Lic82]. This, together with the storage requirement, removes the need for any crossovers. The construction can then follow the design as in Fig. 6. Details are similar to those in [DZ95] and will not be presented. □

[1]This, incidentally, is the actual design of the computer game.

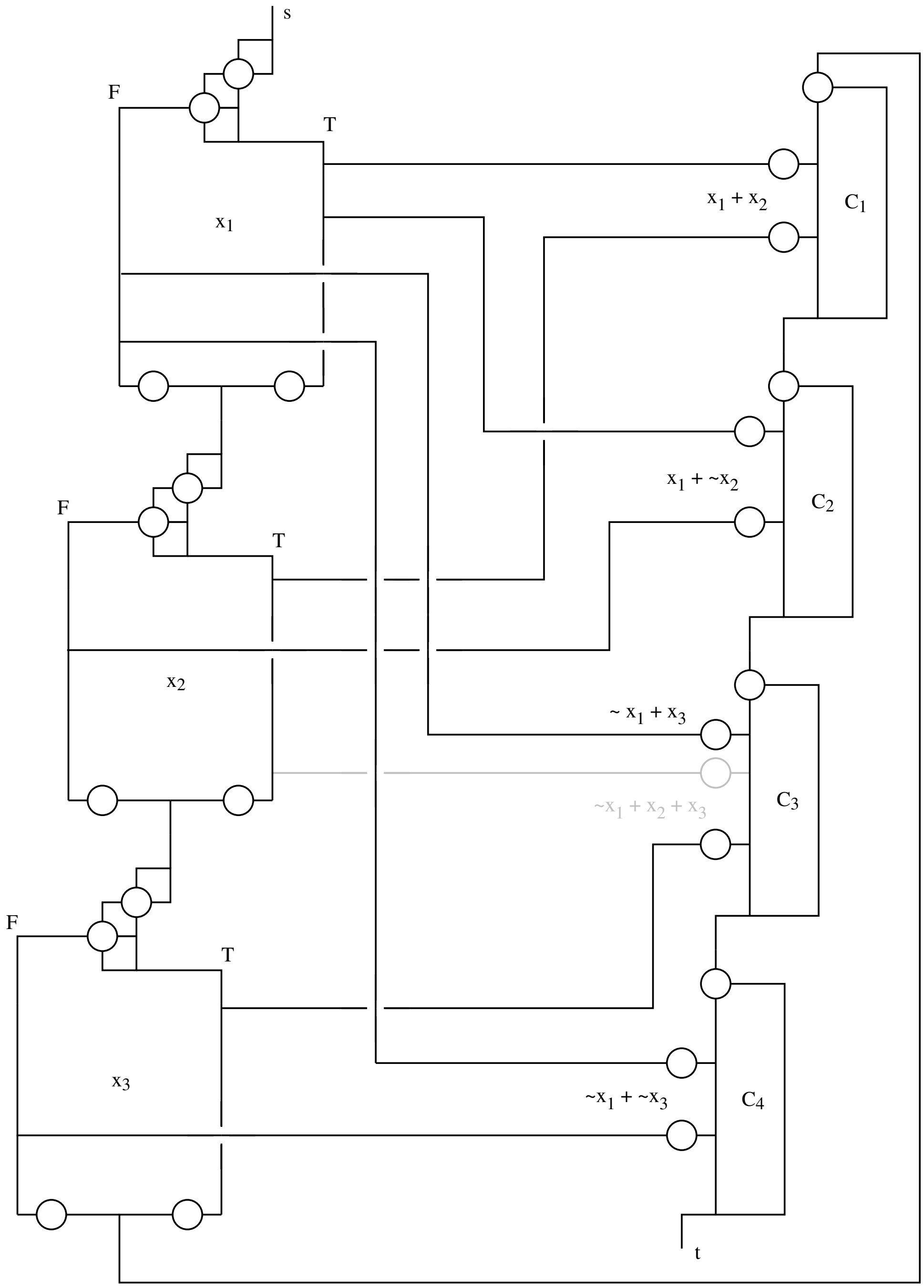


Figure 6: Complete construction for the formulas in Eq. (1) and Eq. (2) (including the shaded portion).

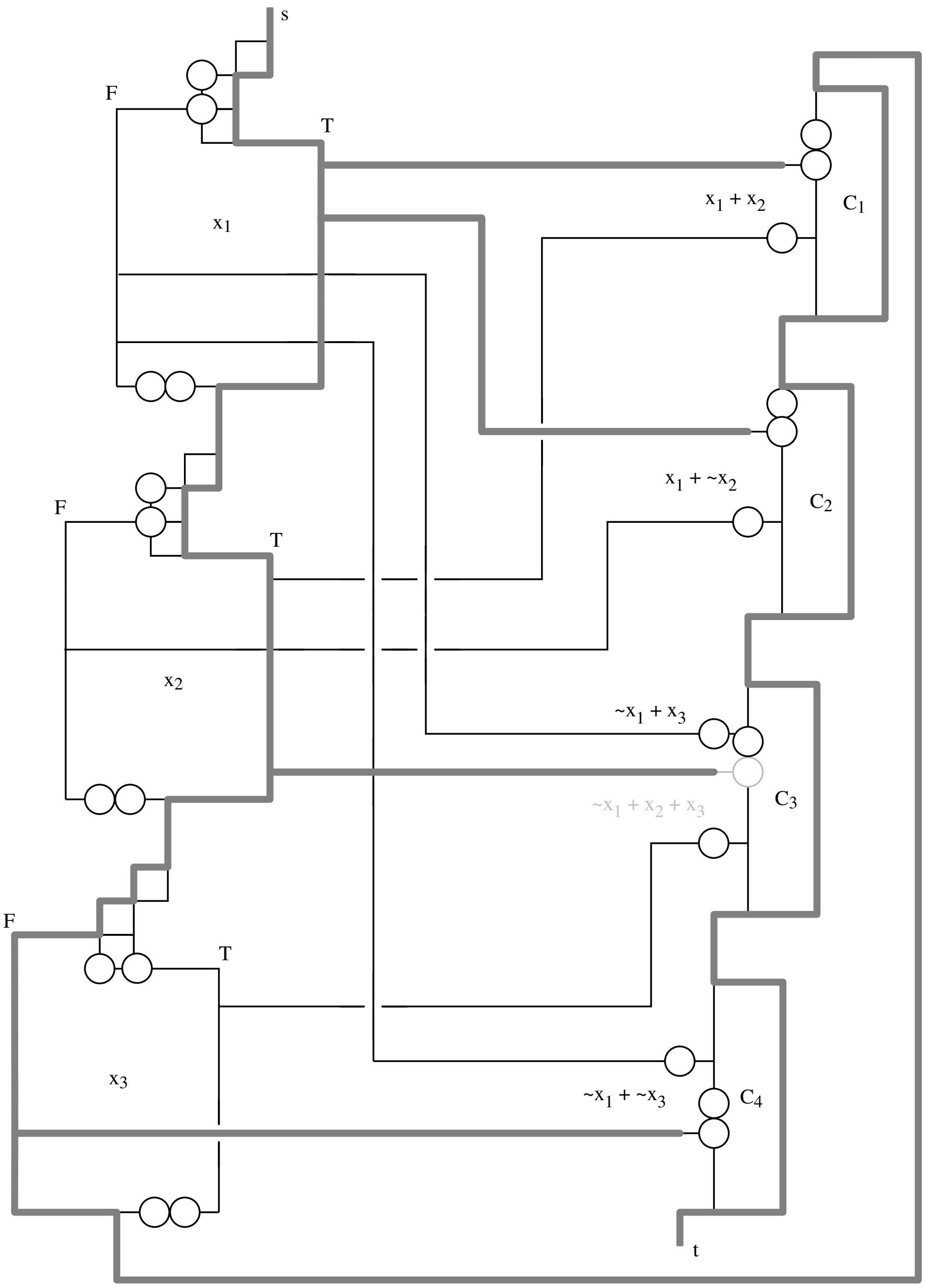


Figure 7: Solution path for Eq. (2).

The reason that Planar 3-SAT does not help for the path version of PushPush is that crossovers are still needed to thread the clause components together into a single path. And it seems that the PushPush conditions are too weak to construct the required crossover gadget:

**Conjecture 1** *No general crossover gadget can be constructed in 2D PushPush.*

Such a gadget would permit two wires to cross, but would prevent leakage from one to the other, just as if it were a 3D crossover. One reason this seems impossible is this:

**Conjecture 2** *No permanent one-way gadget can be constructed in 2D PushPush.*

Note that the the properties of the One-Way gadget in Fig. 1 are destroyed by passage of the robot, after which it becomes a two-way street.

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 this paper. And the last two lines pose two open problems, one raised here, the other in [DO92]: Is PushPush (path version) intractable in 2D? And is the problem where all blocks are movable and the robot can push $k$ blocks, sliding the minimal amount, intractable in 2D?

| 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 |
| push | unit | movable | 1 | storage | *max* | 2D | NP-hard |
| push | unit | movable | 1 | path | *max* | 2D | open |
| push | unit | movable | $k$ | path | min | 2D | open [DO92] |

Table 1: Pushing block problems.

**Acknowledgements.** We thank Erik Demaine, Marty Demaine, and Therese Biedl for helpful discussions.

# 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. Carelton 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.

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