Uniform Robot Relocation is Hard in only two Directions even without Obstacles

David Caballero
University of Texas Rio Grande Valley
Angel A. Cantu
Southwest Research Institute
Timothy Gomez
Massachusetts Institute of Technology
Austin Luchsinger
University of Texas Austin
Robert Schweller
University of Texas Rio Grande Valley
Tim Wylie (timothy.wylie@utrgv.edu)
University of Texas Rio Grande Valley

Research Article

Keywords: Relocation, Swarm Robot Motion Planning, Row Relocation, Tilt Model, Global Uniform Signals

Posted Date: December 21st, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3762289/v1

Additional Declarations: No competing interests reported.

Uniform Robot Relocation is Hard in only two Directions even without Obstacles

David Caballero[1], Angel A. Cantu[2], Timothy Gomez[3], Austin Luchsinger[4], Robert Schweller[1], Tim Wylie[1*†]

[1*]Department of Computer Science, University of Texas Rio Grande Valley, 1201 W. University Dr., Edinburg, TX, 78539, USA.
[2]Southwest Research Institute, 6220 Culebra Road, San Antonio, TX, 78238, USA.
[3]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA, 02139, USA.
[4]Department of Electrical and Computer Engineering, University of Texas Austin, 2501 Speedway, C0803, Austin, TX, 78712, USA.

*Corresponding author(s). E-mail(s): timothy.wylie@utrgv.edu;
Contributing authors: david.caballero01@utrgv.edu; acantu@d16.swri.us; tagomez7@mit.edu; amluchsinger@utexas.edu; robert.schweller@utrgv.edu;
†These authors contributed equally to this work.

Abstract

Given n robots contained within a square grid surrounded by four walls, we ask the question of whether it is possible to move a particular robot a to a specific grid location b by performing a sequence of global *step* operations in which all robots move one grid step in the same cardinal direction (if not blocked by a wall or other blocked robots). We show this problem is NP-complete when restricted to just two directions (south and west). This answers the simplest fundamental problem in uniform global unit tilt swarm robotics. We then consider a relaxed version of this problem in which the goal is to move a robot a to a specific row regardless of its horizontal placement. We show that if asking about the bottom-most row of the square grid, then this version of the problem is solvable in polynomial time. Finally, we discuss several areas for future research and open problems.

Keywords: Relocation, Swarm Robot Motion Planning, Row Relocation, Tilt Model, Global Uniform Signals

1 Introduction

The advanced development of microbots and nanobots has quickly become a significant frontier. However, power and computation limitations at these scales often make autonomous robots infeasible and individually-controlled robots impractical. Thus, recent attention has focused on controlling large numbers of relatively simple robots. Examples of large population robot swarms exist, ranging from naturally occurring magnetotactic bacteria [15–17] to manufactured light-driven nanocars [14, 18]. These swarms of microbots are uniformly manipulated using external stimuli like light, magnetic fields, or gravitational forces. In essence, every agent within the system responds uniformly to the same global signals. This type of global manipulation also reflects the mechanics of many types of systems dating back centuries to marble mazes and other games.

First proposed in 2013 [5], the tilt model consists of movable polyominoes (as an abstraction of these nanorobots) that exist on a 2D grid board with "open" and "blocked" spaces. These polyominoes can be manipulated by a global signal, causing all polyominoes to step a unit distance in the specified direction unless stopped by a blocked space or another polyomino.

Within this model, the complexity of different problems related to the manipulation of the set of polyominoes is studied. The *reconfiguration* problem asks whether one specified configuration is reachable from another by way of these uniform signals. The relocation problem asks whether a specific polyomino or tile can be relocated to a given location (Fig. 1).

Restricted variants of the model are also considered. One of these restrictions is where the polyominoes are limited to single tiles, greatly limiting the complexity of interactions between polyominoes. The other notable restrictions are limiting the global signals to only 2 or 3 directions, and limiting the complexity of the board geometry, i.e., the arrangement of the blocked spaces.

One of the simplest variants of the model is square board geometry, in which the blocked spaces are limited to a square border with no internal geometry, global inputs limited to two directions, and only single tiles. In this simple model, we study the *relocation* problem, showing that the problem of whether a tile can be relocated to a given position is still NP-complete.

1.1 Related Work

Previous research has investigated the manipulation of robot swarms with precise uniform movements in a 2D environment containing obstacles [5]. In the "Full Tilt" variant of this model where tiles slide maximally in each specified direction, the complexity of determining the minimum move sequence for reconfiguration [7], as well as the complexity for Relocation and Reconfiguration [3, 4], have been shown to be PSPACE-complete. Reconfiguration and Relocation have further been shown to be NP-complete when the number of possible directions is limited to 2 or 3 [6]. The single step model, in which robots move a single unit step during each move, was later defined formally, with work studying the complexity of relocating a specified tile to a specific

Problem	Directions	Tile Size	Geometry	Result	Ref.
1^{st} Row Relocation	**2/3**	**1×1**	**Square**	P	**Thm. 2**
Row Relocation	2/3	1×1	Square	open	-
Relocation	**2**	**1×1**	**Square**	**NP-complete**	**Thm. 1**
	2/3	1×1	Monotone	NP-complete	[9]
	4	1×1	General	PSPACE-complete	[10]
	4	1×1,1×2	Square	PSPACE-complete	[10]
Shape Reconfiguration	2/3/4	1×1	Square	NP-hard	[1]
	4	1×1	General	PSPACE-complete	[10]

Table 1: An overview of the complexity results related to the relocation and shape reconfiguration problems in the single step model. The open problems are row relocation in 2 directions in the square and general relocation in the square with four directions. Membership in 4 directions is open for both problems.

location on the board, showing that the problem is PSPACE-complete even when limited to single tiles [10]. The problem of building shapes (adjusting the positions of the robots in the system to collectively form a specified shape) and the problem of building specified patterns out of labelled tiles (i.e. moving the robots into locations such that their labels adhere to a specified shape and pattern) has also been studied, showing that there are board configurations which allow construction of general shapes in optimal time [11] and patterned shapes in near-optimal time [8].

Previous work has also studied restrictions on this model. The two main restrictions studied are limiting the number of directions the robots can move in, and limiting the complexity of the board's geometry. A hierarchy of board geometries is described in [4]. It was shown that when limiting the number of available directions to 2 and with "monotone" board geometry the problem of relocation is NP-complete [9].

The simplest variant of the model, in which there are single tiles in a square board with no internal obstacles, has not been studied extensively. When all four directions are allowed, work has shown that the problem of arranging the robots into a specific shape is NP-hard [1]. Depending on the starting configuration, the tiles can be compacted in an exponential number of ways. When the tiles get compacted, they form a permutation group that was studied in detail in [2]. However, the complexity for relocation and reconfiguration with four directions is still an open question.

1.2 Contributions

We investigate the relocation problem in the single step model. Table 1 shows what was previously known and how our results relate. We answer an open question about the simplest version of the problem. We show that relocation when limited to single tiles, only two directions, and no blocking geometry is still NP-complete. With this in mind, we have also shown that knowing whether a tile can be relocated to the bottom row is in P [12], however, whether a tile can reach an arbitrary row is still an open problem.

We first overview the unit movement (or single step) tilt model in Section 2. In Section 3 we show that with two directions in the square, relocation is NP-complete. Finally, we overview the first-row relocation algorithm in Section 4, along with the difficulties that arise when moving to general row relocation. Finally, several important open problems are outlined in the conclusion (Section 5).

This work is an extension of a conference version with the hardness proof [13], and of a 2-page abstract outlining the first row relocation problem [12]. The journal has extended these in several ways with full details and proofs of the positive result as well as additional future work and open problems.

2 Preliminaries

We give the model and problem definitions related to single step tilt in an open board.

Board. A *board* (or *workspace*) is a rectangular region of the 2D square lattice in which specific locations are marked as *blocked*. Formally, an $m \times n$ board is a partition $\mathbb{B} = (O, X)$ of $\{(x, y) | x \in \{1, 2, \dots, m\}, y \in \{1, 2, \dots, n\}\}$ where O denotes a set of *open* locations, and X denotes a set of *blocked* locations- referred to as "concrete." Here, we use the most restrictive geometry in the hierarchy where O is a square and the only blocked locations are the edges around the board.

Tiles. A tile is a unit square centered on a non-blocked point on a given board. Formally a tile t stores a coordinate on the board c and is said to occupy c.

Configurations. A configuration is an arrangement of tiles on a board such that there are no overlaps among tiles, or with blocked board spaces. Formally, a configuration $C = (\mathbb{B}, \mathbb{T} = \{t_1, \dots, t_k\})$ consists of a board $\mathbb{B}$ and a set of non-overlapping tiles $\mathbb{T}$. We say two configurations $C = (\mathbb{B}, \mathbb{T} = \{t_1, \dots, t_k\})$ and $C' = (\mathbb{B}, \mathbb{T}' = \{t'_1, \dots, t'_k\})$ have the same shape if $\mathbb{T}$ and $\mathbb{T}'$ are translations of each other. The shape of a configuration C is the shape of $\mathbb{T}$.

Step. A *step* is a way to turn one configuration into another by way of a global signal that moves all tiles in a configuration one unit in a direction $d \in \{N, E, S, W\}$ when possible without causing an overlap with a blocked position, or another tile. Formally, for a configuration $C = (\mathbb{B}, \mathbb{T})$, let $\mathbb{T}'$ be the maximal subset of $\mathbb{T}$ such that translation of all tiles in $\mathbb{T}'$ by 1 unit in the direction d induces no overlap with blocked squares or other tiles. A step in direction d is performed by executing the translation of all tiles in $\mathbb{T}'$ by 1 unit in that direction.

We say that a configuration C can be *directly reconfigured* into configuration C' (denoted $C \rightarrow_1 C'$) if applying one step in some direction $d \in \{N, E, S, W\}$ to C results in C'. We define the relation $\rightarrow_*$ to be the transitive closure of $\rightarrow_1$ and say that C can be *reconfigured* into C' if and only if $C \rightarrow_* C'$, i.e., C may be reconfigured into C' by way of a sequence of step transformations.

Step Sequence. A *step sequence* is a series of steps which can be inferred from a series of directions $D = \langle d_1, d_2, \dots, d_k \rangle$; each $d_i \in D$ implies a step in that direction. For simplicity, when discussing a step sequence, we just refer to the series of directions from which that sequence was derived. Given a starting configuration, a step sequence

(a) Initial (b) $\langle S \rangle$ (c) $\langle W \rangle$ (d) $\langle W \rangle$

Fig. 1: An example step sequence. The initial board configuration followed by the resulting configurations after an $\langle S \rangle$ step, $\langle W \rangle$ step, and then final $\langle W \rangle$ step. The red tile is the one to relocate and the red outline square is the target location.

corresponds to a sequence of configurations based on the step transformation. An example step sequence $\langle S, W, W \rangle$ and the corresponding sequence of configurations can be seen in Fig. 1.

Relocation. Given a tilt system with an $n \times n$ board $\mathbb{B}$, a set of tiles $\mathbb{T} = \{t_1, \dots, t_m\}$ where each $t_l = (x, y)$ s.t. $1 \leq l \leq m$, $1 \leq x \leq n$, and $1 \leq y \leq n$. Given $t_i, t_j \in \mathbb{T}$, $t_i \neq t_j$ if $i \neq j$. For shorthand, we use $t_{i,j}$ for $t_l = (i, j)$. For the row or column, we use t_{l_r} and t_{l_c}.

Given a specific tile to relocate t_R at location $(r, c) = (t_{R_r}, t_{R_c})$, and a target location $T = (T_r, T_c)$, the relocation problem asks whether a series of steps can translate t_R s.t. $(t_{R_r}, t_{R_c}) = T$.

Definition 1 (Knitting)**.** *The* knitting row *and* knitting column *are the row and column of t_R. Knitting is the act of performing $\langle W \rangle$ movements (or $\langle S \rangle$) when every position of the knitting area (row or column) is occupied by a tile. Thus, t_R maintains its position.*

3 2-Direction NP-Hard Relocation

We design gadgets that encode truth values of literals for a given 3SAT instance equation. We provide two step-sequences for 'assigning' truth values to variables, which reconfigure the gadgets into two distinct configurations. A 'true' value for a literal is interpreted as the presence of an 'output' tile within a target location in a gadget, whereas a 'false' value is simply the absence of that tile. We group three gadgets together to create a clause and check for 3SAT satisfiability by counting the number of output tiles in the gadgets after assigning truth values to all variables. We show that relocation becomes impossible if step-sequences are used beside the ones provided, giving us strict control over the outcome of the system. The move directions considered henceforth are $\langle S, W \rangle$.

Layout.

Given a 3SAT instance, we construct a board divided into three regions called the equation section, relocation section, and helper section (Figure 2). The equation section is composed of multiple subregions called *clause* spaces each containing three gadgets assigned to the three literals for that particular clause. The helper section is the region next to the equation section that contains floating tiles used to generate

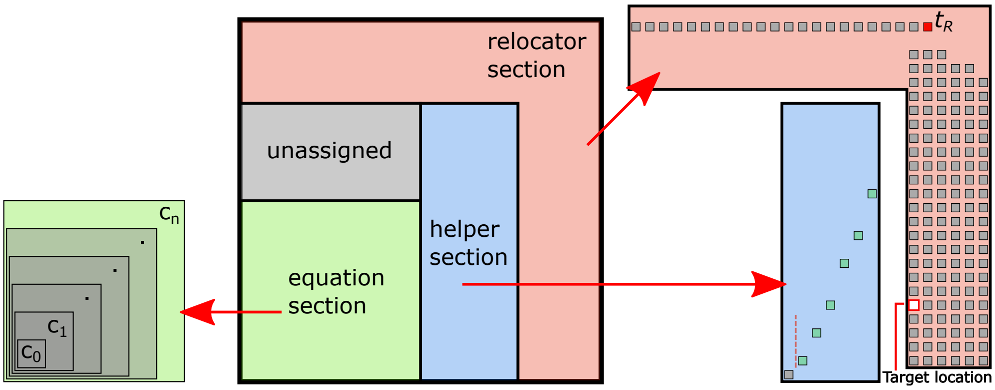
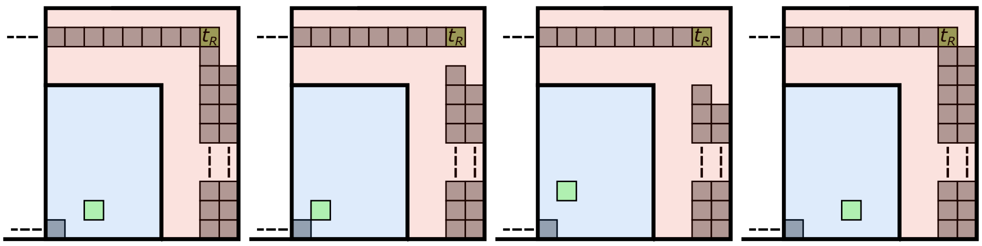


Fig. 2: A high-level view of the layout of the gadgets on the board.



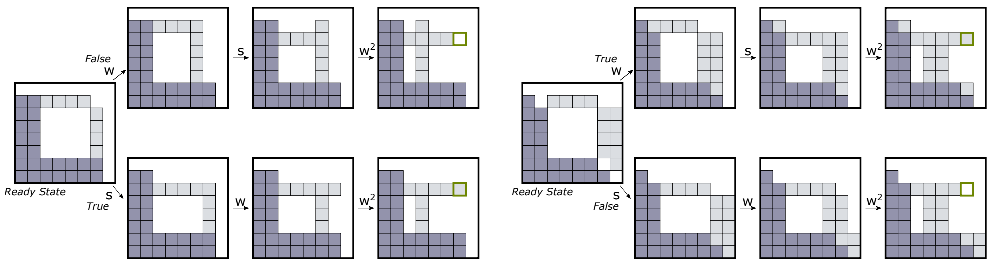
(a) $\langle \mathbf{w}, \mathbf{s} \rangle$ (b) $\langle \mathbf{s}, \mathbf{w} \rangle$ (c) $\langle \mathbf{s}, \mathbf{s} \rangle$ (d) $\langle \mathbf{w}, \mathbf{w} \rangle$

Fig. 3: Dashed lines represent tiles that extend to the edges of the board. The figures show how different combinations of helper tile's position (green tile) and t_R's position can generate any forced movement sequence. Each example represents possible positionings of the helper tile and t_R such that the listed move sequence is forced, given that the helper tile must be placed at the bottom of the board and t_R remain adjacent to the row of tiles.

forced step-sequences. As detailed in Lemma 1, each helper tile must reside at the bottom of the board in order to geometrically assist the target tile for relocation. The relocator section consists of a row of tiles extending from the left edge of the board along with multiple columns of tiles underneath it that extend from the bottom edge of the board. The target tile $t_R = (t_{R_r}, t_{R_c})$ is defined as the last tile of the row in the relocator section with target location $T = (T_r, T_c)$ such that $T_c = t_{R_c} - 1$ and $T_r = |C| + 2$ for a set of clauses C.

Force Moves.

Forcing a step-sequence is achieved by purposely preventing relocation if that sequence is not used. This is done by either trapping the target tile via geometric blocking or preventing the target tile from interacting with other tiles needed for relocation. The columns in the relocator section and helper tiles in the helper section are used



(a) Literal Gadget

(b) Negated Literal Gadget

Fig. 4: The gadgets are in the *ready* state at the beginning of each sequence. Both gadgets are depicted along with how each 'assign' step-sequences affect them. We can track a gadget's truth value by observing the length of the horizontal pillar such that it is assigned true if the pillar is lengthened by a single tile after a step-sequence.

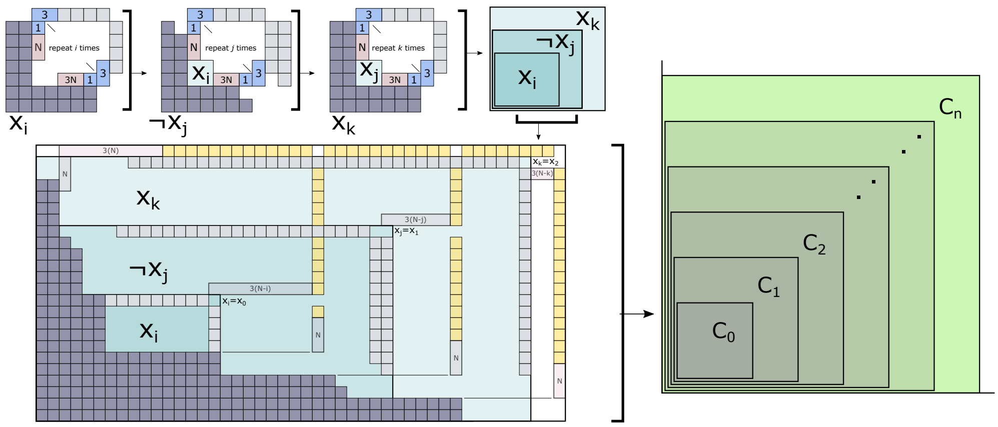
together to generate any forced step-sequence (Figure 3). A $\langle W \rangle$ move is forced when the target tile resides just above a column in the relocator section since a $\langle S \rangle$ move pushes the row of tiles next to the target tile downwards, making the target tile stuck above the column. A $\langle S \rangle$ move is forced when a $\langle W \rangle$ move places a helper tile in the same column as another helper tile, therefore making it impossible to place all helper tiles at the bottom of the board.

Lemma 1. *Every helper tile in the helper section must be placed at the bottom edge of the board in order to make relocation of the target tile possible.*

Proof. The row of tiles adjacent to the target tile prevents it from stepping into the column before it (moving the tile west), blocking the target tile from entering the column of the target location. The target tile must eventually break away from the row by moving on a column of tiles, pushing the row downwards, and moving west into the column before it. We make this scenario available only once when we check if every clause of the 3SAT equation is satisfied and stack as many tiles as there are satisfied clauses beneath the target tile. Similarly, placing every helper tile at the bottom of the board creates a row of tiles just long enough to occupy a position in the same column as the target tile. By positioning the target location $|C| + 2$ above the bottom of the board, every clause must be satisfied along with every single helper tile in the helper section placed at the bottom of the board. The additional tile in the equation comes from a tile we initialize on the board for the purpose of functionality. If a single helper tile is not placed at the bottom of the board, the row of helper tiles can not be long enough to occupy a position in the column of the target tile in the disengage part of the reduction, therefore relocation becomes impossible. □

Gadgets.

Gadgets are composed of 'pillars' of tiles that extend from blocked tiles adjacent to the bottom and left edges of the equation section. We provide two versions of a gadget for normal and negated literals shown in Figure 4. We define two 'assign' step-sequences: 'assign true' as $\langle s, \mathbf{w}, \mathbf{w}, \mathbf{w} \rangle$ and 'assign false' as $\langle w, \mathbf{s}, \mathbf{w}, \mathbf{w} \rangle$ such that the last three



(a) Nested Gadgets and Clauses

Fig. 5: (a) Depiction of clause $c = (x_0 \vee \neg x_1 \vee x_2)$ with $N = 3$ distinct variables. Gadgets and clauses are nested inside each other in order to prevent unwanted intervention of their components.

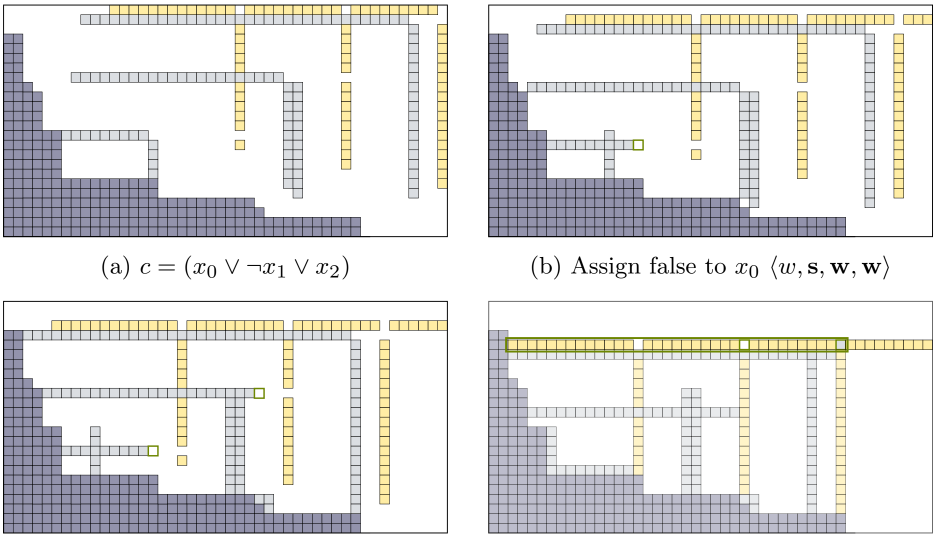
moves are forced. In the reduction, we execute one of the two 'assign' step-sequences for every variable of a given 3SAT equation so that each gadget assigned to a literal encodes the truth value of that literal in the length of the horizontal pillar. That is, a gadget (literal) evaluates to true if the horizontal pillar lengthens by one after a 'assign' step-sequence is used or false if the pillar remains the same length. The *output position* of a gadget is defined as the position on the horizontal pillar that contains, or does not contain, the additional tile after the 'assign' step-sequence. For the gadgets assigned to literals x_i and x_j where $i < j$, we space out the pillars of x_j so that when x_i is in the *ready* state (see Figure 4), the pillars of x_j are $\langle s^{1\times j}, w^{3\times j}\rangle$ spaces away from the *ready* state. This allows us to assign truth values to each variable in order independently of each other.

Clause Spaces.

For the set of clauses C of a given 3SAT instance, we define the clause space for clause $c_i \in C$ as the region on the board with three gadgets assigned to each literal in $c_i = (x_i, x_j, x_k)$. The gadgets are allocated consecutively such that the 'next' gadget encompasses the 'previous' gadget by lengthening its pillars with dimensions detailed in Figure 5. We similarly build each clause space such that the 'next' clause space encompasses the 'previous' clause space as shown in Figure 6. With this design, each clause space functions independently and in parallel with the other clause spaces.

System Output.

Given a sequence of truth assignments for the variables, determining if a clause was satisfied involves placing as many tiles on a single row in the clause space, called the *clause output*, as there are satisfied literals in the clause. To do this, we position floating



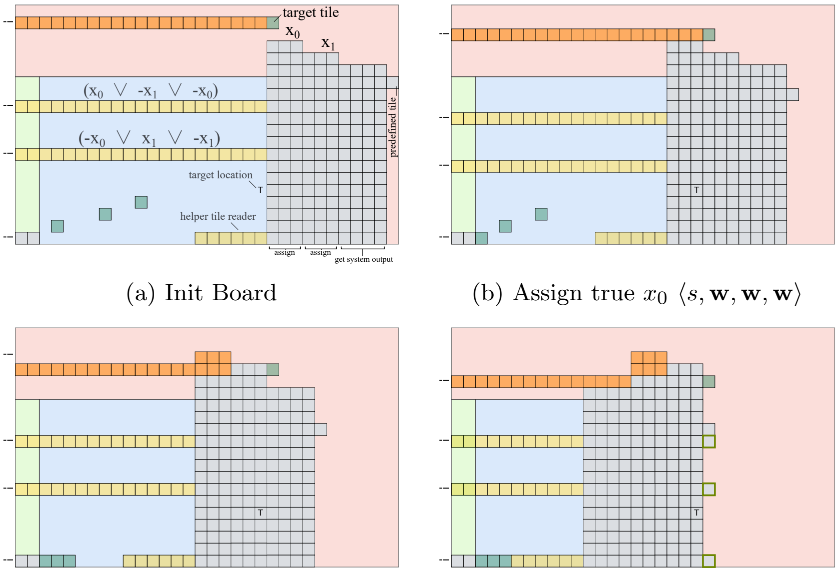
(a) $c = (x_0 \vee \neg x_1 \vee x_2)$ (b) Assign false to x_0 $\langle w, \mathbf{s}, \mathbf{w}, \mathbf{w} \rangle$

(c) Assign true to x_1 $\langle s, \mathbf{w}, \mathbf{w}, \mathbf{w} \rangle$ (d) Assign true to x_2

Fig. 6: Given clause $c = (x_0 \vee \neg x_1 \vee x_2)$ with distinct variables $N = 3$. (a) We first assign 'false' to x_0, which makes literal gadget x_1 move to the *ready* state. (c-d) We then assign false and true to x_1 and x_2, respectively, followed by pushing the output tiles of the gadget to the clause output row.

columns of tiles called *readers* that wrap around each gadget output position after the last variable truth assignment based on the dimensions given in Figure 5. As shown in Figure 6, this allows us to step south and lengthen the reader by a single tile if the literal evaluates to true. If at least one reader is lengthened by one, then the clause is said to be satisfied given that the reader occupies a position in the clause output.

To determine satisfiability of the 3SAT equation, we position horizontal readers that extend from the relocator section which wrap around each clause output after using the first readers, seen in Figure 6d. By repeatedly moving west, these readers are compressed and push out a tile in the relocator section for every satisfied literal in the clause as shown seen Figure 7d. This way, when the target tile reaches the last column of the relocator section, the amount of tiles underneath the target tile is at least the number of satisfied clauses. Similarly, we utilize a reader for the helper tiles in order to join the two rows and occupy a position underneath the target tile given that every helper tile is present in the row. If every clause is satisfied, and every helper tile is placed at the bottom edge of the board, then the target tile can 'disengage' with the row of tiles next to it by stepping downwards until compressing with the tiles beneath it and then relocate to the target location. Similarly, if at least one clause is unsatisfied, or at least one helper tile was not placed at the bottom edge of the board, then the target tile can not 'disengage' with the row of tiles in the same row as the target location, and therefore can not relocate. With this, we define the *get system output* sequence as $\langle s, w, w, w, w \rangle$.



(a) Init Board

(b) Assign true x_0 $\langle s, \mathbf{w}, \mathbf{w}, \mathbf{w} \rangle$

(c) Assign true x_1 $\langle s, \mathbf{w}, \mathbf{w}, \mathbf{w} \rangle$

(d) Get System Output

Fig. 7: The sections are not to size and for demonstrative purposes only. (a) Example of the right side of the board with clauses $(x_0 \vee \neg x_1 \vee \neg x_0)$ and $(\neg x_0 \vee x_1 \vee \neg x_1)$ and variables $N = 2$. The first six columns of tiles in the relocation section, together with the first two helper tiles, generates two assign step-sequences for the variables. (b-c) Assigning true to both variables makes the 3SAT equation evaluate to true. The last helper tile and four columns of tiles in the relocator section forces the user to compress the readers and push out a tile underneath the target tile per satisfied literal.

Lemma 2. *Single Step Relocation in a square board with only two directions is NP-hard.*

Proof. We prove this by a reduction from 3SAT. Given a 3SAT instance, we construct a board divided into three sections called the equation section, helper section, and relocator section. From Lemma 1, we can generated any 'forced' step-sequence by utilizing helper tiles in the helper section and columns in the relocator section to create scenarios in which an incorrect step-sequence results in the impossibility of relocation. With this capability, we design two step-sequences for assigning truth values to each distinct variable in the 3SAT equation. We force N of any of these two step-sequences at the beginning of the reduction so that each step-sequence reconfigures the appropriate gadgets, where N is the number of distinct variables of the 3SAT instance. Next, we execute the *get system output* step-sequence, which involves moving readers around gadget outputs in order to push out as many tiles as there are satisfied literals to a single row within a clause region. This is followed by a second group of readers that wrap around clause region outputs and push out a tile in the relocation section, underneath the target tile, if a particular clause is satisfied. Afterwards, the

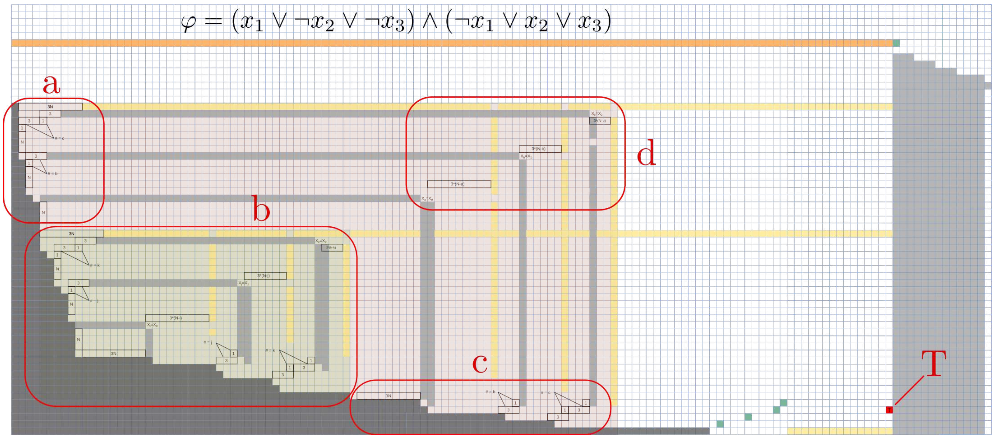


Fig. 8: Initial board example for 3SAT equation $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$. Section details are shown in Figures 9 and 10.

satisfiability of the 3SAT equation is evaluated to true if the number of tiles underneath the target tile equals to $|C| + 2$. We get $|C|$ tiles if each clause was satisfied and 1 tile from the helper tiles. The last tile is automatically given since it is pre-initialized on the board. We can see that if any of these conditions are not met, then relocation is impossible. That is, relocation of the target tile is possible if and only if every clause is satisfied and each helper tile is placed at the bottom of the board. □

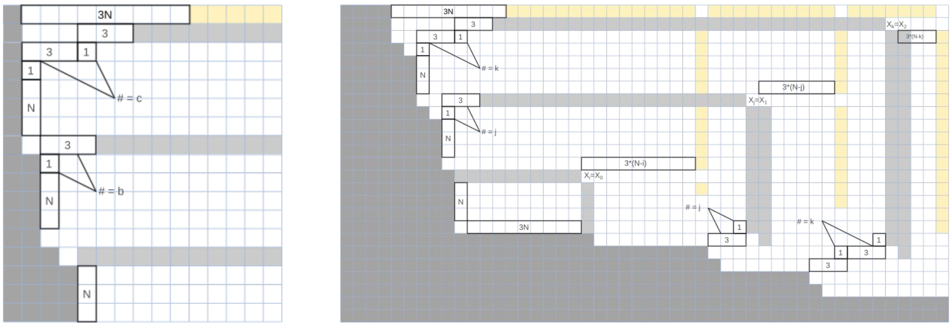
Relocation Membership.

Membership in NP for the particular instance we are considering subtly depends on the problem definition and encoding. The single step tilt model, as defined, is a set of open and blocked spaces. Thus, the set of tiles is a subset of those locations, and membership in NP is straightforward. This was shown in [9]. However, given the nature of the square board with all spaces open, an alternate formulation of this specific variant of the problem could take in the dimension of the board, n, encoded in binary, which would imply the board size is exponential in the input size. Each tile can be encoded as only its starting location, which can also be encoded in binary. Such an input would mean the obvious certificate for relocate-ability would no longer be polynomial sized. Membership in NP is still an open question for this version of the problem.

Lemma 3. *Single Step Relocation in a square board with only two directions is in NP [9].*

Theorem 1. *Single Step Relocation in a square board with only two directions is NP-complete.*

Proof. Follows from Lemmas 2 and 3. □



(a) Section (a) from Fig. 8

(b) Section (b) from Fig. 8

Fig. 9: Sections (a) and (b) of Figure 8. In (a), we depict how many spaces the horizontal pillar of each literal gadget is from the ready state. The horizontal reader is spaced out by $3N$ in order to account for each variable assignment. (b) The dimensions for each gadget in the lowest clause space is depicted.

4 Row Relocation

Given that the general relocation problem in the square is hard even with only two directions, we know relax the problem a bit to solve something more general. Here, we look at *row relocation*, which asks if we can move some tile t_R to a specific row r regardless of its column position. We show that *first row relocation* has a polynomial-time solution.

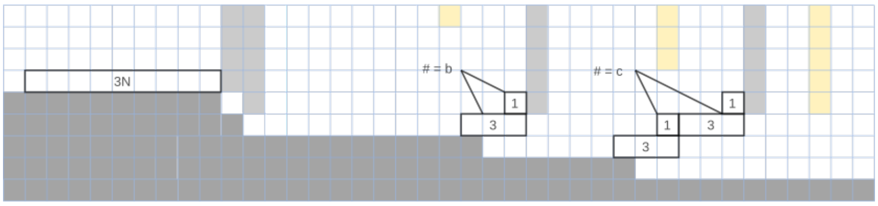
To successfully relocate tile t_R to the bottom row, it is crucial to ensure that no other tiles are positioned beneath it. This involves identifying an appropriate empty column. Subsequently, we use a technique called *knitting*, which facilitates the movement of the empty column directly below t_R. Additionally, we label the board into distinct sections (refer to Figure 11a). Our strategy includes locating E_c, the empty column, and assigning *counts* to the rows. These counts represent the number of tiles influencing the relocation process. For visual examples of the empty column and the associated counts, see Figure 11b.

4.1 Formal Definitions

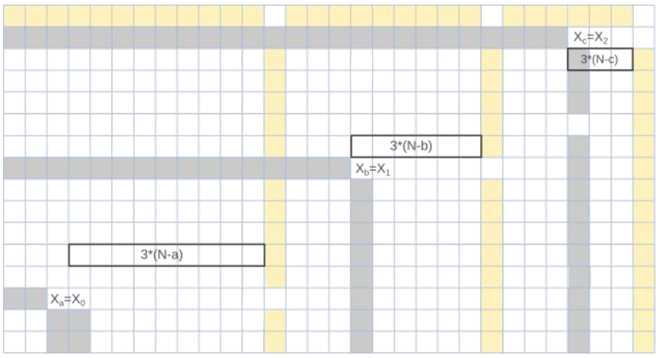
We quickly overview the concepts needed for our algorithm. We have attempted to make the section fairly independent, and thus some of the definitions are repeated for convenience.

Definition 2 (Row Relocation)**.** *Given a specific tile to relocate* t_R *at location* $(r, c) = (t_{R_r}, t_{R_c})$*, and a target row* T_r*, the row relocation problem asks whether a series of transformations can translate* t_R *s.t.* $t_{R_r} = T_r$*.*

Definition 3 (First Row Relocation)**.** *Given a specific tile to relocate* t_R *at location* $(r, c) = (t_{R_r}, t_{R_c})$*, the first row relocation problem asks whether a series of transformations can translate* t_R *s.t.* $t_{R_r} = 1$*.*



(a) Section (c) from Fig. 8



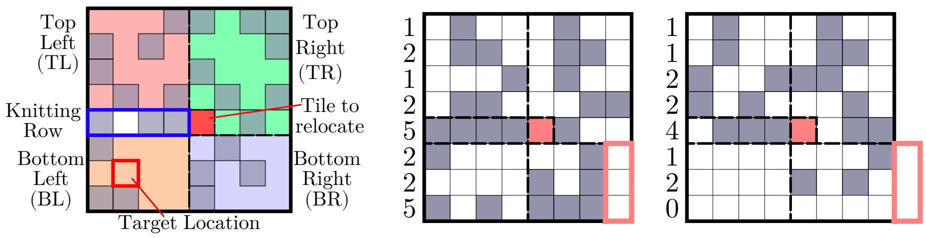
(b) Section (d) from Fig. 8

Fig. 10: Sections (c) and (d) of Figure 8. (c) Similarly, each vertical pillar is spaced out given the dimensions depicted. The lower literal gadget in the clause space is provided with enough horizontal space to allow for all variable assignment step-sequences to occur without interference from other tiles on the board. (d) The vertical readers' dimensions for the upper clause space is depicted.

Definition 4 (Knitting)**.** *The row between the BL and TL section is the knitting row. Knitting is the act of performing* $\langle W\rangle$ *movements when every position of the knitting area is occupied by a tile. Thus,* t_R *maintains its position.*

Definition 5 (Empty Column E_c)**.** *Given a tilt system board configuration* $\mathbb{S} = (\mathbb{B}, \mathbb{T})$ *and tile* $t_R = (r, c)$ *where* $t_R \in \mathbb{T}$*, define* $E_c = \min\{k : c \le k \le n+1 \ s.t.\ |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i < r, j = k\}| = 0\}$*. If all columns in the BR section have tiles,* $E_c = n + 1$ *and enters the board after a west,* $\langle W\rangle$*, movement.*

Definition 6 (Counts)**.** *Given a tilt system* $\mathbb{S} = (\mathbb{B}, \mathbb{T})$*, where* $\mathbb{B}$ *is an* $n \times n$ *grid and* $\mathbb{T}$ *is a set of tiles each with a location in the grid, and a tile* $t_R \in \mathbb{T}$ *with location* (r, c)*,*



(a) Board Sections (b) Example counts

Fig. 11: (a) The board is an $n \times n$ square divided into 6 sections based on the tile to relocate. The four large sections are the Top Left (TL), Top Right (TR), Bottom Left (BL), and Bottom Right (BR) sections. There is also the knitting area (outlined in blue) and the tile to relocate (red dot). When we move to general relocation, there will also be a target spot (red square). (b) Some configurations with the counts for each row listed to the left and E_c highlighted.

and the target empty column E_c. Define the count of a row k as follows.

$$Count(\mathbb{T}, k, t_{r,c}) = \begin{cases} |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i = k, 1 \leq j < c\}|, & \\ \qquad \textit{if } k > r \ (\textit{rows above } t_R) & \\ |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i = k, 1 \leq j \leq c\}|, & \\ \qquad \textit{if } k = r \ (t_R \textit{ row}), & \\ |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i = k, 1 \leq j < E_c\}|, & \\ \qquad \textit{if } k < r \ (\textit{rows below } t_R) & \end{cases}$$

Definition 7 (Candidate Row)**.** *Given a tilt system board configuration $\mathbb{S} = (\mathbb{B}, \mathbb{T})$ and a tile $t_R \in \mathbb{T}$ at location (r, c). The knitting row may contain up to $c - 1$ tiles. The closest row (fewest south, $\langle S \rangle$, movements) in the TL section with a count higher than the knitting row is the candidate row. Let k be the number of tiles in the knitting area, $k = |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i = r, j < c\}|$. Then the current candidate row (CR) is*

$$CanRow(\mathbb{T}, t_{r,c}) = \min\{i : i > r, Count(\mathbb{T}, i, t_{r,c}) > k\}.$$

CR is the row index, but for notational convenience, we let $|CR|$ be the count of the candidate row: $|CR| = Count(\mathbb{T}, CR, t_{r,c})$.

Definition 8 (Empty Column E_c)**.** *Given a tilt system $\mathbb{S} = (\mathbb{B}, \mathbb{T})$ and tile $t_R = (r, c)$ where $t_R \in \mathbb{T}$, define $E_c = \min\{k : c \leq k \leq n + 1 \ s.t. \ |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i < r, j = k\}| = 0\}$.*

Definition 9 (Counts)**.** *Given a tilt system* $\mathbb{S} = (\mathbb{B}, \mathbb{T})$*, a tile* $t_R = (r, c)$ *where* $t_R \in \mathbb{T}$*, and* E_c*, Define the count of a row* k *as follows.*

$$Count(\mathbb{T}, k, t_{r,c}) = \begin{cases} |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i = k, 1 \le j < c\}|, & \textit{if } k > r \textit{ (above } t_R\textit{)} \\ |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i = k, 1 \le j \le c\}|, & \textit{if } k = r \textit{ (} t_R \textit{ row),} \\ |\{t_{i,j} : t_{i,j} \in \mathbb{T}, i = k, 1 \le j < E_c\}|, & \textit{if } k < r \textit{ (below } t_R\textit{)} \end{cases}$$

This basic framework leads to two important lemmas (4, 5). Algorithm 1 ensures that one of these is eventually met.

4.2 Existence Properties

Using the counts and the knitting area, we have the following two lemmas.

Lemma 4. *[Existence] If the count of the knitting area is greater than the counts of the lower section, then first row relocation is possible.*

Proof. Given that the counts in the bottom section correspond to the number of tiles extending up to the empty column E_c, we infer that when E_c is positioned under the tile targeted for relocation, the space to the left of every spot in the empty column is at least as large as the highest count in the lower sections. This arrangement ensures that each tile in the bottom section can fit to the left of the empty column if we position the empty column beneath the relocation tile. Therefore, this configuration allows for the relocation of the tile.

□

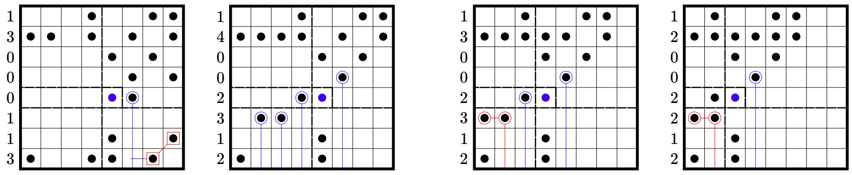
Lemma 5. *[Nonexistence] If there exist a count in the bottom sections that is larger than every count in the TL section and knitting area, then first row relocation is impossible.*

Proof. Assume that the empty column may be translated beneath the relocation tile. Moreover, consider the best case where the largest count in the TL section resides in the knitting area. Therefore, by applying a $\langle W \rangle$ movement, the relocation tile remains at the same location on the board. If the empty column is translated beneath the relocation tile, the area to the left of every position in the empty column is less than the largest count in the bottom sections. Thus, at least one tile in the bottom sections must reside in the empty column, which contradicts the assumption that the empty column is movable beneath the relocation tile. □

4.3 Changing the Knitting Row

If neither condition in the lemmas is satisfied, we cannot knit with the tiles in the knitting row. Define the *candidate row* as the closest row in the TL section with a count higher than the knitting row.

Definition 10 (Candidate Row)**.** *Given a tilt system board configuration* $\mathbb{S} = (\mathbb{B}, \mathbb{T})$ *and a tile* $t_R \in \mathbb{T}$ *at location* (r, c)*. The knitting row may contain up to* $c - 1$ *tiles. The closest row (fewest south,* $\langle S \rangle$*, movements) in the TL section with a count higher than the knitting row is the candidate row.*



(a) Tiles in H (b) Reducing H

Fig. 12: (a) Tiles that enter the BR section after D movements are included in N_R. If the empty column is occupied by a tile after D movement (i.e., it becomes non-empty), then every tile between the empty column and the next empty column (shown in red) are also added to N_R. Reducing the number of tiles in H is done by $\langle W \rangle$ movement. (b) Tiles in H that lead towards another tile beneath it are removed from H.

If we make any movements, we may introduce new tiles that must be considered. We look at new tiles that may enter the BL section (N_L) and tiles that may enter in the BR section (N_R). Define the number of $\langle S \rangle$ movements needed for the candidate row to enter the knitting area as D. This includes $\langle S \rangle$ movements needed to get the relocation tile adjacent to tiles in its column. Thus, for Figure 12a, $D = 4$ since the candidate row with count 3, is 3 away from the knitting area, but another $\langle S \rangle$ move is needed before the knitting area also stops moving south. Essentially, in order to make the Candidate Row be on the same row as the tile to relocate, we also need to do some vertical knitting.

Let H be the set of all tiles in either the TL or BL section that are within D $\langle S \rangle$ movements from the target row. These are all tiles under the Candidate Row that could be relocated to the target row. Formal analysis of introducing tiles into N_L or N_R is shown below.

4.4 Analysis of N_L

Tiles in N_R from Moving the Candidate Row. There are two ways we may add tiles to N_R: tiles entering through $\langle S \rangle$ movements, and tiles that enter because the first empty column changes (E_c) due to new tiles. We include the set of tiles in TR that are at most D distance (with $\langle S \rangle$ moves) from the BR that increase the counts of the bottom rows (Figure 12a). Note that the BR and TR sections may change if t_R moves. If E_c has tiles after D movements, then we change E_c to the correct column. Now all tiles that enter the last row after D movements to the left of the new empty column are included in N_R.

4.5 Analysis of N_R

Tiles in N_L from Moving the Candidate Row. Let N_L be the tiles that enter the BL section through the TL section after D $\langle S \rangle$ movements. However, we may be able to reduce this number. Let H be the set of all tiles in either the TL or BL section that are within D $\langle S \rangle$ movements from the target row. These are all tiles under the Candidate Row that could be relocated to the target row. See Figure 12b for an example showing tiles in H that may be removed from H by making $\langle W\rangle$ movements before the $\langle S\rangle$ movements.

For a tile in H to be stacked on a tile below, the tile below would have to be adjacent to the wall to another stationary tile, and thus it remains in its position after $\langle W\rangle$ movement. With this in mind, we make the following observation.

Lemma 6. *If the inclusion of N_R in the counts of the lower rows causes any row to exceed $|CR|$, then relocation is impossible.*

Proof. We prove this by arguing the impossibility of diminishing $|N_R|$. Trivially, the number of tiles in N_R does not diminish with $\langle S\rangle$ movement. Now, consider a tile $t_{i,j} \in N_R$ that arrives at the last row through the open space located directly below it on the first row. To have this tile not arrive at the open space after D movements, it must land on another tile (stacking) to the right or left of the open space. If by some number of $\langle W\rangle$ movements, the tile leads towards a tile to the *right* of the open space, it follows that the row of tile $t_{i,j}$ is maximized, and therefore the tile $t_{i,j-1}$ leads towards the location of the open space instead, thus not reducing the number of paths. On the other hand, if after some $\langle W\rangle$ movement, the tile leads towards a tile to the *left* of the location the open space, then it follows that the row to the left of the open space is maximized, and thus relocation is impossible (Lemma 5). Therefore, it is impossible to reduce $|N_R|$. □

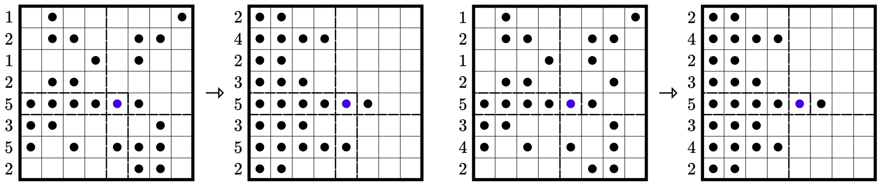
Lemma 7. *To reduce $|H|$, it suffices to perform $\langle W\rangle$ movements rather than some combination of $\langle S\rangle$ and $\langle W\rangle$ movements.*

Proof. Consider tile $t_{i,j} \in H$ and tile $t_{i-1,j-1}$ that is adjacent to the wall. If tile $t_{i,j}$ does not reside above $t_{i-1,j-1}$ when a $\langle W\rangle$ movement is performed, then there must exist a tile $t_{i-1,j}$ that is to the left of $t_{i,j}$. Note that there will always exist a tile to the left of $t_{i,j}$ regardless of the number of $\langle S\rangle$ movement performed. Thus, if tile $t_{i,j}$'s path to the last row could not have been removed by a $\langle W\rangle$ movement, then neither can it be removed with any combination of $\langle W\rangle$ and $\langle S\rangle$ movements. The only manner that its path to the last row can altered is with tiles residing in the rows beneath it and in columns to the right of it, which can be moved underneath it with only $\langle W\rangle$ movements. □

4.6 First Row Relocation Algorithm

When we attempt to get the candidate row tiles on the same row as the tile to relocate, in order to do knitting, we must consider new tiles added to the BL and BR sections. If the counts are greater than the count of the candidate row, then we are not able to get t_R to the target row. However, some of the tiles in N_L might also be in H and can possibly be stacked in another column. Thus, if the current CR can not be used due to the counts, we can try to reduce H and N_L by $\langle W\rangle$ movements. Thus, we iteratively attempt this until either we find a solution, or the counts show it is impossible. If there are larger candidate rows, we can also attempt a larger one (a higher count).

When first row relocation is possible, the sequence is always $W^u S^v W^x S^y$ where u, v, x, y must be determined, but are bounded by n. There are $\leq c-2$ possible candidate rows, and at most $n-r$ moves to bring any CR to the knitting row.



(a) Has no solution (b) Knitting solution

Fig. 13: (a) Two examples of configurations where first row relocation is not possible. (b) Two configuration examples where first row relocation is possible via knitting.

Algorithm 1: First Row Relocation

Data: $\mathbb{S} = (\mathbb{B}, \mathbb{T})$, where $\mathbb{B}$ is an $n \times n$ grid and $\mathbb{T}$ is a set of tiles each with a location in the grid, and a tile $t_{r,c} \in \mathbb{T}$ with location (r, c).
Result: Is First Row Relocation possible?
$CR = r$; (Set CR to be the knitting row.)
while *Lemmas 4 and 5 are unsatisfied* **do**
    /*Is it currently possible?*/
    Determine E_c and counts for each row;
    if *Lemma 4 is satisfied* **then** accept;
    if *Lemma 5 is satisfied* **then** reject;
    /*Find and move a candidate row*/
    Set candidate row CR;
    if $\nexists$ *another candidate row* **then** reject;
    Calculate D, N_R, N_L, E_c, and new counts;
    if *D $\langle S \rangle$ movements satisfy Lemma 5* **then** reject (Lemma 6);
    /*Can H be reduced?*/
    Determine H;
    if *Lemma 4 is satisfied* **then** accept;
    else if *Lemma 5 is satisfied* **then** break;
    else Perform $\langle W \rangle$ movements until H changes (Lemma 7);
if *Lemma 4 is satisfied* **then** accept;
if *Lemma 5 is satisfied* **then** reject;

Theorem 2. *First row relocation of tile $t_{r,c} \in \mathbb{T}$ on an $n \times n$ board can be solved in $\mathcal{O}(cn + c|\mathbb{T}|)$ time.*

Proof. The proof is that first row relocation is solvable by Algorithm 1. Essentially, through $\langle W \rangle$ or $\langle S \rangle$ movements, Either Lemma 4 or 5 must eventually be true. By Lemma 6, there are only so many $\langle S \rangle$ movements that can be made, and by Lemma 7, it suffices to only make $\langle W \rangle$ movements to try and reduce tiles in H that might add to the counts. Thus, any successful sequence is always $W^u S^v W^x S^y$ where u, v, x, y are bounded by n, and there are less than c possible candidate rows. Algorithm 1 guarantees they are tried in succession.

Complexity: Determining the empty column, E_c takes $\mathcal{O}(\mathbb{T})$ time to look at all tiles. Counting takes $\mathcal{O}(\mathbb{T})$ time given a linked list implementation where we have a list of tiles sorted by row and column. There are c possible candidate rows which each may require up to n W movements to check if they work. Thus, the total number of steps is at least $\mathcal{O}(cn)$. However, for each tile in the TL section, we might need to change the counts. Thus $\mathcal{O}(c|\mathbb{T}|)$ is required because we need to redo the counts for each candidate row. □

5 Conclusion

In this work we answered an open question by showing that relocation in the single step tilt model, even in the most restrictive case with no fixed geometry except the borders of the space and with only two movement directions, is still NP-complete. As shown in Table 1, there is now a fairly complete characterization of this problem in relation to movement direction, tile size, and board geometry. A few important questions remain, which we overview here.

- Is the relocation problem in the square in NP if the input is specified as the tile locations and a binary encoded integer for the board size? As mentioned, membership is not obvious since the number of steps needed may be exponential in the size of the input.
- In the square with four directions, is single step relocation or shape configuration in NP? Recent work by [2] outlined the basic permutation groups that occur in a polyomino under the single step model, but there is no work addressing the compaction of tiles into different permutation groups. It may be that relocation is not in NP because an exponential number of moves is necessary to move a tile into the correct permutation group, move it to the correct spot in a shape, and then move the shape in the square.
- Following from the previous question, the same reasoning is why membership is still open for reconfiguration (and why all results in Table 1 are only NP-hard). Is shape reconfiguration in the square in NP?
- For the single step tilt model in the square, is general row relocation in P or is it still NP-hard? In [12], they show that knowing whether a tile can relocate to the bottom row (1^{st} Row Relocation) is in P. It is fairly straightforward to modify the 1^{st} Row Relocation algorithm to work for the 2^{nd} row, but every additional row seems to add a higher polynomial. It is clearly bounded by the number of alternations needed between W and S. If only k alternations are needed, then row relocation is possible in $\mathcal{O}(n^k)$, giving a poor FPT algorithm. Can the number of alternations be bounded by the problem instance for a better FPT?
- Following from the previous question, is there a configuration requiring $\mathcal{O}(n)$ alternations between W and S?

Declarations

Ethical Approval

Not applicable.

Competing interests

There are no competing interests that we are aware of in reference to this paper.

Authors' contributions

These authors contributed equally to this work.

Funding

No external funding was received.

Availability of data and materials

Data Availability Statement: No Data associated in the manuscript.

References

[1] Akitaya H, Aloupis G, Löffler M, et al (2016) Trash compaction. In: Proc. 32nd European Workshop on Computational Geometry, pp 107–110

[2] Akitaya HA, Löffler M, Viglietta G (2022) Pushing blocks by sweeping lines. In: Proc. of the 11th Inter. Conf. on Fun with Algorithms, FUN'22

[3] Balanza-Martinez J, Luchsinger A, Caballero D, et al (2019) Full tilt: Universal constructors for general shapes with uniform external forces. In: Proc. of the 2019 ACM-SIAM Symposium on Discrete Algorithms, SODA'19, pp 2689–2708, https://doi.org/10.1137/1.9781611975482.167

[4] Balanza-Martinez J, Gomez T, Caballero D, et al (2020) Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces. In: Proc. of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA'20, pp 2625–2641, https://doi.org/10.1137/1.9781611975994.160

[5] Becker AT, Habibi G, Werfel J, et al (2013) Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 520–527, https://doi.org/10.1109/IROS.2013.6696401

[6] Becker AT, Demaine ED, Fekete SP, et al (2014) Reconfiguring massive particle swarms with limited, global control. In: Algorithms for Sensor Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 51–66

[7] Becker AT, Demaine ED, Fekete SP, et al (2014) Particle computation: Designing worlds to control robot swarms with only global signals. In: IEEE International Conference on Robotics and Automation, ICRA'14, pp 6751–6756, https://doi.org/10.1109/ICRA.2014.6907856

[8] Caballero D, Cantu AA, Gomez T, et al (2020) Building patterned shapes in robot swarms with uniform control signals. In: Proceedings of the 32nd Canadian Conference on Computational Geometry, CCCG'20, pp 59–62

[9] Caballero D, Cantu AA, Gomez T, et al (2020) Hardness of reconfiguring robot swarms with uniform external control in limited directions. Journal of Information Processing 28:782–790

[10] Caballero D, Cantu AA, Gomez T, et al (2020) Relocating units in robot swarms with uniform control signals is pspace-complete. In: Proceedings of the 32nd Canadian Conference on Computational Geometry, CCCG'20, pp 49–55

[11] Caballero D, Cantu A, Gomez T, et al (2021) Fast reconfiguration of robot swarms with uniform control signals. Natural Computing 20:1–11. https://doi.org/10.1007/s11047-021-09864-0

[12] Caballero D, Cantu AA, Gomez T, et al (2021) Unit tilt row relocation in a square (short abstract). In: Proceedings of the 23rd Thailand-Japan Conference on Discrete and Computational Geometry, Graphs, and Games, TJCDCG3'2020+1, pp 122–123

[13] Caballero D, Cantu AA, Gomez T, et al (2023) Uniform robot relocation is hard in only two directions even without obstacles. In: Unconventional Computation and Natural Computation. Springer Nature Switzerland, Cham, UCNC'23, pp 17–31

[14] Chiang PT, Mielke J, Godoy J, et al (2012) Toward a light-driven motorized nanocar: Synthesis and initial imaging of single molecules. ACS Nano 6(1):592–597. https://doi.org/10.1021/nn203969b, pMID: 22129498

[15] Felfoul O, Mohammadi M, Gaboury L, et al (2011) Tumor targeting by computer controlled guidance of magnetotactic bacteria acting like autonomous microrobots. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 1304–1308, https://doi.org/10.1109/IROS.2011.6094991

[16] Martel S (2012) Bacterial microsystems and microrobots. In: Biomedical Microdevices, pp 1033–1045, https://doi.org/10.1007/s10544-012-9696-x

[17] Martel S, Taherkhani S, Tabrizian M, et al (2014) Computer 3d controlled bacterial transports and aggregations of microbial adhered nano-components. Journal of Micro-Bio Robotics 9(1):23–28. https://doi.org/10.1007/s12213-014-0076-x

[18] Shirai Y, Osgood AJ, Zhao Y, et al (2005) Directional control in thermally driven single-molecule nanocars. Nano Letters 5(11):2330–2334. https://doi.org/10.1021/nl051915k, pMID: 16277478