Hardness of Traversing Gadget Systems with Small Bandwidth

MIT Gadgets Group
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Erik D. Demaine
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Jenny Diomidova
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Timothy Gomez
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Markus Hecher
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Jayson Lynch
Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

The motion-planning-through-gadgets framework has enabled proofs of PSPACE-completeness for many motion-planning problems, ranging from swarm and modular robotics to DNA computing to video games. In this paper, we strengthen this framework to show that, for several useful gadgets and gadget families, motion planning remains PSPACE-complete even when gadgets are connected together into a graph of constant bandwidth (which implies constant pathwidth, treewidth, and cliquewidth). We then show how this result applies to several geometric/grid-based motion-planning problems, establishing PSPACE-completeness even when restricted to a rectangle/box where only one dimension is large (superconstant). On the positive side, we find one family of gadgets (DAG gadgets) for which motion planning is fixed-parameter tractable with respect to bandwidth.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Mathematics of computing → Graph algorithms; Theory of computation → Abstract machines; Theory of computation → Turing machines

Keywords and phrases Gadgets, Motion Planning, Parameterized Complexity, Hardness

Digital Object Identifier 10.4230/LIPIcs.SAND.2025.11

Funding This research was funded by the Austrian Science Fund (FWF), grants J 4656 and P 32830, as well as the Society for Research Funding in Lower Austria (GFF) grant ExzF-0004.

1 Introduction

Whenever considering a new model of computation, a fundamental first question is whether the model is **universal**: is it as powerful as possible as all models with the same resource constraints? Typically, models restrict either the amount of time or the amount of space/states allowed, in which case universality corresponds to NP-completeness or PSPACE-completeness respectively. Such hardness results are usually proved using a reduction that consists of several constant-size constructions (called "gadgets" or "gates") and an algorithm for connecting them together to represent an arbitrary computation.

When the model consists of dynamic moving element(s) in a graph or geometry, we say that it involves **motion planning**, and then a particularly useful framework for proving such hardness results is the **motion-planning-through-gadgets** framework [16, 17, 8, 7, 9, 12, 11, 10]. In general, this framework aims to characterize which gadget(s) (typically, just one gadget) suffice to prove NP- or PSPACE-completeness of a given motion-planning problem, provided copies of those gadget(s) can be constructed and wired together to form an arbitrary graph. This framework has been successfully applied to prove PSPACE-completeness of many geometric motion-planning problems, arising in diverse areas such as swarm robotics with uniform global control [15], programmable matter through modular robots [1], DNA computing through Chemical Reaction Networks [3] even on a surface [2], as well as video games [4, 8, 6] and related puzzles [7, 9, 5].

Motion-planning gadgets are thus the foundation of a web of reductions that is rapidly growing, and thus is a great target for deeper investigations in terms of structural graph parameters and parameterized complexity. For example, many motion-planning-through-gadgets results remain hard when the graph of gadget connections is planar, which is useful for many 2D applications. What other graph properties can be imposed while still maintaining the model's universality? Additionally what non-structural parameters can we impose on gadgets which model important parameters in the motivating problems, one example is fixing the number of global state changes or the number of times the agent changes its environment.

1.1 Our Results

In this paper, we first consider the **bandwidth** of the graph of connections between gadgets, and show that reachability for several previously studied gadgets and gadget families remain PSPACE-complete when restricted to constant-bandwidth graphs (which also implies constant pathwidth, treewidth, and cliquewidth). **Bounded-bandwidth graphs** are particularly important for geometric applications, as such graphs can be embedded into a constant-height grid. Thus, as applications of this bounded-bandwidth framework, we obtain PSPACE-hardness for many problems in a constant-height rectangle (or in higher dimensions, a box with only one superconstant dimension). Table 1 summarizes these results.

We start in Section 3 by presenting a 4-state gadget for which the motion-planning problem[1] is PSPACE-complete even with bandwidth 7. In parameterized complexity terminology, we prove that motion planning is **paraPSPACE-complete** with respect to bandwidth. This result implies that motion planning with any universal gadget, such as doors [8] and certain I/O gadgets [12], is paraPSPACE-complete with respect to bandwidth. The I/O results hold even in zero-player settings, where there are no choices to be made but rather the problem is to predict the behavior of the system after a specified amount of time. In Section 4, we show that a well-studied family of gadgets – reversible deterministic gadgets – also inherit paraPSPACE-complete results with respect to bandwidth.

While most gadget problems appear to be hard for small-bandwidth graphs, we do show one exception. In Section 5, we show that **DAG gadgets** [17] – a well-studied family of gadgets in the class NP – are not only polynomial for constant bandwidth and gadget size,

[1] Specifically, we consider the problem of **agent reachability**, which asks whether an agent can reach a given location in a given system of gadgets.

Table 1 Our results characterizing the complexity of motion planning (both reachability and reconfiguration problems) with small (constant or fixed-parameter) bandwidth. * 3-way branch is only required for 1-player results.

Players	Gadget (Family)	Parameter	Result	Cite
0/1-Player	TM-cell	Bandwidth-7	PSPACE-c	Thm. 1, 3
0/1-Player	doors, 3-way branch*	Bandwidth-214	PSPACE-c	Cor. 2
0/1-Player	I/O gadgets, 3-way branch*	Bandwidth-$\mathcal{O}(1)$	PSPACE-c	Cor. 4
1-Player	locking 2-toggle, 3-way branch	Bandwidth-$\mathcal{O}(1)$	PSPACE-c	Thm. 8
1-Player	interacting-k-tunnel rev.det., 3-way branch	Bandwidth-$\mathcal{O}(k)$	PSPACE-c	Cor. 9
1-Player	DAGs, 3-way branch	Fixed Treewidth + States	FPT	Thm. 10
1-Player	interacting-k-tunnel LDAGs, 3-way branch	Fixed Global State Change	W[SAT]-hard, XP	Thm. 11

Table 2 Application problems for which we show hardness holds even for constant-height rectangles, implied by our results from Table 1 and by [22].

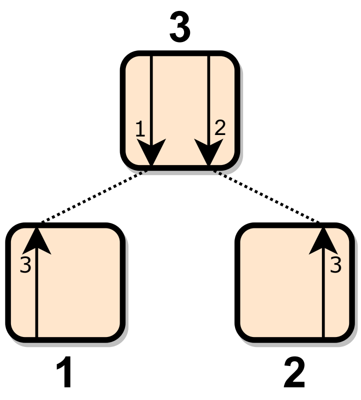
Model, Puzzle, Game	Complexity	Reduces From	Cite
Surface CRN 1-Reconfiguration	PSPACE-c	Locking 2-Toggle	[3]
Tilt on a Rectangular Board	PSPACE-c	Crossing Toggle Lock	[15]
Modular pivoting robots reconfiguration	PSPACE-c	Protected Locking 2-Toggle	[1]
PushPull-1F	PSPACE-c	Crossing Locking 2-Toggle	[17]
Super Mario 64	PSPACE-c	Symmetric Self-Closing Door	[8]
Super Mario World	PSPACE-c	Directed Door Opening	[6]
Sokobond	PSPACE-c	Door Gadgets	[8]
Donkey Kong Country 1, 2, 3	PSPACE-c	Door Gadgets	[4]
The Legend of Zelda: A Link to the Past	PSPACE-c	Door Gadgets	[4]
The Legend of Zelda: Oracle of Seasons	PSPACE-c	Crossing Toggle Lock	[16]

but are in fact **fixed-parameter tractable (FPT)** with respect to the number of states and treewidth (and hence bandwidth). For non-structural parameters we study fixing the number of global state changes. We give a family of 2-state gadgets for which Reachability is W[SAT]-hard with respect to number of state changing traversals.

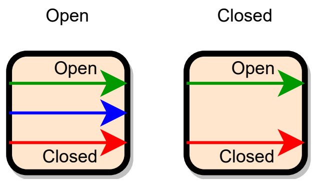
Studying parameterized versions, such as bandwidth, of problems in the gadget framework gives us **parameterized hardness**, such as height, for the models that implement gadgets. Here we point out a few examples of results implied by our reductions and reductions in [22], summarized in Table 2. Many of these reduce from universal gadgets, or Nondeterministic Constraint Logic.

2 Gadget Model

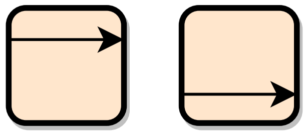
A **gadget** consists of ℓ **locations**, s **states**, and a set of allowable **transitions** $(s, l) \to (s', l')$ among state–location pairs. Such a transition indicates that, when the gadget is in state s, the agent can enter at location l and exit at location l', changing the gadget's state to s'. A **system of gadgets** consists of a set of gadgets, their initial states, and a **connection graph** which connects locations of gadgets in a matching. To connect together more than two locations, we require a "branching-hallway" gadget which is a 1-state 3-location gadget where all traversals are possible. The **reachability** problem asks, given a system of gadgets, a start location for the agent, and a goal location, can the agent make legal traversals to reach the goal location? The **reconfiguration** problem asks, given a start location for the agent and a target configuration of the system, can the agent traverse the gadgets to reach the target configuration?



(a) The locking 2-toggle gadget (L2T). In the top state 3, you can traverse either tunnel going down, which blocks off the other tunnel until you reverse the initial traversal.



(b) A directed open-required door.



(c) A symmetric self-closing door.

Figure 1 Gadgets commonly used for reductions in other models.

We discuss several special classes of gadgets in this paper and collect those definitions here for convenience.

- A **branching hallway** is a gadget that has a single state and three locations. All traversals are always allowed. This models choices for the agent.
- An **input/output** gadget can have its locations partitioned into two sets, "inputs" and "outputs", such that all transitions go from input locations to output locations.
- A **DAG gadget** is a gadget where no state can be repeated (so the induced graph on states is a DAG), or equivalently, can be traversed only a finite number of times.
- A **reversible** gadget satisfies that, for every transition $(X, o) \to (X', o')$, there is also the reverse transition $(X', o') \to (X, o)$.
- A **deterministic** gadget is one in which every state–location pair (X, o) has at most one transition from it.
- A **tunnel** gadget has a partial matching on the locations such that all transitions only connect locations in that matching. A k-tunnel gadget has k of these pairs.
- An **interacting-k-tunnel gadget** is a k-tunnel gadget where there exists a transition that changes whether another tunnel in the gadget can be traversed. See [17] for a more complete definition.
- A **door gadget** is a gadget with an open tunnel, a traverse tunnel, and a close tunnel. The gadget has two states, determining whether the traverse tunnel is open (traversable) or closed (untraversable). Going through the open or close tunnel sets the state respectively. In a **directed door**, each tunnel can be traversed in only one direction.
- A **symmetric self-closing door** is a gadget with two tunnels and two states. The state determines which of the tunnels is open (traversable). Going through the tunnel changes the state, thus closing the tunnel you just traversed and opening the opposite tunnel.

Both gadgets in Figure 1 are interacting-k-tunnel gadgets. The L2T (Figure 1a) is reversible and deterministic. The directed door (Figure 1b) is an *input/output gadget.*

Structural Parameters of Gadgets

For studying (graph) parameters on a system of gadgets, we need a suitable graph representation. There are multiple ways one can define such **representations of a system of gadgets**. One can take each gadget and replace it by a single vertex with edges for all the connections from any location of that gadget. Alternatively one can replace each gadget with a graph that has vertices for each location of the gadget and edges based on all transitions between locations. We believe this is likely the best to represent the complexity that occurs in reductions using the gadgets framework, but we instead use the following definition, as it is simpler than the transition definition and provides an upper bound on these other versions. We define the **structure graph** of a system of gadgets, which is the graph obtained from the connection graph by replacing each k-location gadget with a k-clique with a node at each location.

Having the structure graph representation, we briefly recall the definition of bandwidth and treewidth. Let $G = (V, E)$ be a graph and $f : V \to \{1, \ldots, |V|\}$ be a *bijective mapping* that uniquely assigns a vertex to an integer from 1 to the number of vertices. The *(graph) dilation* of G and f is the maximum (absolute) difference between integers assigned to adjacent vertices which we call the length of the edge, i.e., $\max_{\{u,v\}\in E} |f(u) - f(v)|$. The **bandwidth** of G is the minimum dilation of G among every bijective mapping f for G.

A *tree decomposition (TD)* of G is a pair $\mathcal{T} = (T, \chi)$ where T is a tree, and χ is a mapping that assigns to each node t in T a set $\chi(t) \subseteq V$, called a *bag*, such that two conditions hold: (i) "covered": $V = \bigcup_{t' \text{ in } T} \chi(t')$ and $E \subseteq \bigcup_{t' \text{ in } T}\{\{u, v\} \mid u, v \in \chi(t')\}$; (ii) "connected": for each r, s, t such that s lies on the path from r to t, we have $\chi(r) \cap \chi(t) \subseteq \chi(s)$. The *width* of $\mathcal{T}$ corresponds to $\max_{t' \text{ in } T} |\chi(t')| - 1$. The **treewidth** of G is the minimum width over all tree decompositions of G.

3 Turing Machine Gadgets

We show there exists a gadget for which motion planning is hard even with a bandwidth of 7. We reduce from the problem of "Given a **Linear Bounded Automata (LBA)** does the head ever point to the rightmost cell?" This can be shown hard by reducing from the **halting problem** for LBAs [13]. Add an additional cell to the right of the LBA with a special symbol. We can assume the original LBA does not move to this cell since we can mark the original rightmost cell with a special symbol. Once the LBA halts we can have the head move all the way rightward ignoring the special symbol to reach the target cell. This works even in the case of universal Turing machines since an LBA is a special case of a Turing machine which is promised to never go beyond its allotted tape size. Thus an UTM can simulate this while never leaving the prescribed tape section.

A $\mathbb{TM}$-cell gadget is a 4-state 8-location input/output gadget with 16 transitions. We base our gadget on the 2-state 4-symbol **weakly universal Turing machine (UTM)** in [19] and draw the state diagram in Figure 2. We label the states of the UTM u_1 and u_2. Each gadget represents a cell of the tape. The symbol stored in a cell is represented by the state of the gadget. Each gadget has four input locations and four output locations. Both sets of locations are labeled with the cross product of $\{u_1, u_2\}$ and $\{\text{Left}, \text{Right}\}$. If the head in state u_1 moves right into a cell it will enter from location u_1, "Right", In. The transitions of the gadget are based on the instructions of the UTM, meaning a robot that enters a gadget in the state for 0 through a location representing state u_1 will leave through location u_1, Left, Out and change the gadget state to !0. This models the UTM head being in state u_1 reading a 0, then writing a !0 to the tape and moving left in state u_1. The output location for state u_1 on the right side connects to the input location for state u_1 on the left side of the adjacent gadget. This arrangement can be seen in Figure 3. Weakly universal requires that a periodic pattern is repeated on both sides of the tape. We set this pattern by setting the initial configuration of the gadgets to the states representing this pattern.

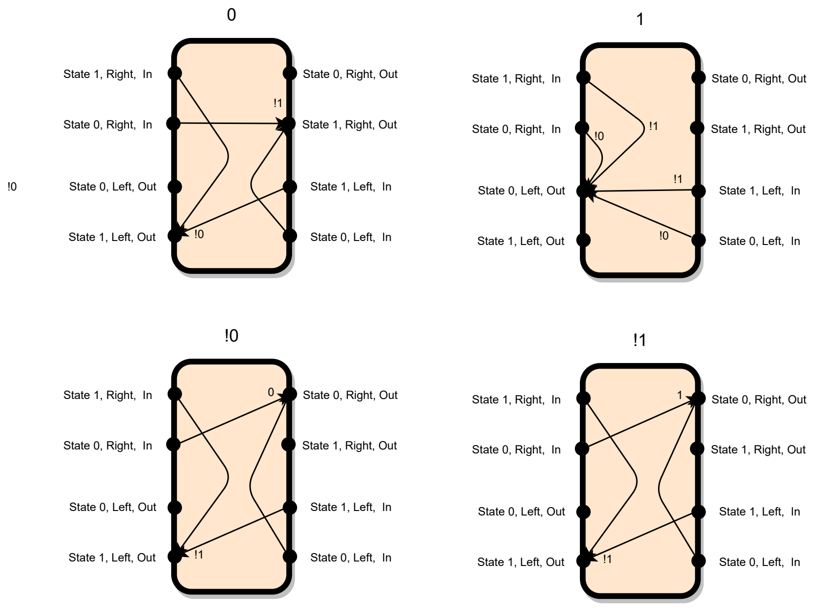


Figure 2 The states of the 𝕋𝕄-cell gadget. Each gadget state describes a different symbol written on the cell.

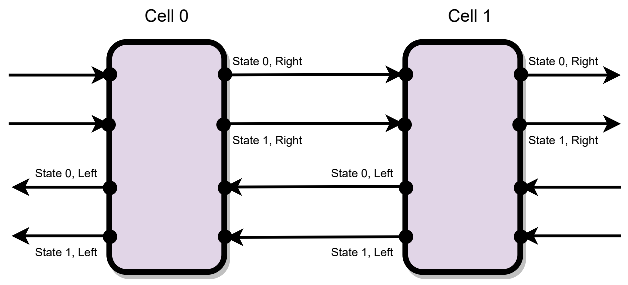


Figure 3 Each 𝕋𝕄-cell gadget represents one cell in the Turing Machine tape. They are connected in a line as in the diagram.

▶ **Theorem 1.** *Motion planning for the* 𝕋𝕄*-cell gadget is PSPACE-complete even with bandwidth* 7*.*

Proof. The 𝕋𝕄-cell gadget has 8 ports so the bandwidth of these nodes is the bandwidth of a complete graph of size 8 which is 7. Two neighboring gadgets can be drawn such that edges of the connection graph are all length 3.

We reduce from the problem described above by replacing the rightmost cell with the target location. The robot only has the option to move forward since the gadgets are input/output gadgets and the reduction does not have any branching gadgets. If the head ever reaches the rightmost cell the robot is able to reach the target. ◀

Next, we use the notion of universal gadgets to derive more results. A **universal gadget** can be used to simulate any other gadget. Door gadgets [8] as well as the unbounded multi-input output-disjoint deterministic 2-state input/output gadgets [12]. Both of these simulations use a number of gadgets which is a function of the states and locations of the gadget to be simulated. Since that is a constant the simulation only increases the bandwidth by a constant factor. For a gadget with s states, ℓ locations, and t transitions it can be simulated with $2\ell + s + t$ doors. An example of a door gadget can be seen in Figure 1b. Each door has six locations. This gives an upper bound of 28 doors and we can calculate the bandwidth of this simulation which is $12\ell + 6s + 6t - 2 = 214$, which can probably be improved significantly with optimization of the simulation and a more careful examination of the graph structure of the simulation. For the planar door gadgets which can construct crossovers, we can even have bounded bandwidth for planar motion planning, given some additional blowup due to the additional need to simulate crossovers.

▶ **Corollary 2.** *Motion planning for doors is PSPACE-complete even with* 214 *bandwidth.*

3.1 0-Player Motion Planning and I/O Gadgets

As noted in the definition, the $\mathbb{TM}$-cell gadget is an input/output (I/O) gadget. This requires that the locations can be split into two disjoint sets where one set only serves as inputs and the other as outputs. Here we use this fact to achieve additional results. First we note the Turing machine construction works directly in the 0-player I/O model [12]. We then give another corollary for universal I/O gadgets.

The **0-player motion planning model** is described in [12]. We give a simplified definition here. A system of I/O gadgets is a valid 0-player system if each output location only connects to a single input location in the connection graph. The agent then cannot make any choices and the system behaves more as a simulation.

▶ **Theorem 3.** *0-player motion planning for the $\mathbb{TM}$-cell gadget is PSPACE-complete even with bandwidth* 7*.*

Proof. In our connection graph each location only connects to a single neighbor and all trainsitions go from input locations to output locations. Thus there were no choices for the agent to make and the 0-player and 1-player versions of the problem are equivalent. ◀

More information on universal I/O gadgets along with some examples can be found in [12].

▶ **Corollary 4.** *0-player and 1-player motion planning for the set of universal I/O gadgets is PSPACE-complete even with $O(1)$ bandwidth.*

3.2 Consequences for Other Parameters

Due to known graph parameter relations [21], we immediately obtain the following.

▶ **Corollary 5.** *Motion planning for the $\mathbb{TM}$-cell gadget is PSPACE-complete even with $\mathcal{O}(1)$ pathwidth, treewidth, or cliquewidth.*

Since the location graph for fixed gadgets has constant degree, we also obtain the following.

▶ **Corollary 6.** *Motion planning for the $\mathbb{TM}$-cell is PSPACE-complete even with $\mathcal{O}(1)$ cutwidth.*

Proof. The cutwidth cw of a graph is bounded by $\Delta \cdot bw$; the degree Δ is constant and by Theorem 8, constant bw suffices for hardness. ◀

3.3 More Turing Machine Gadgets

We also mention the method for encoding a universal Turing Machine in a gadget can be generalized. Thus any improvements in this area would imply improvements in these reductions. There cannot be a 1-state universal Turing machine, so this won't improve the bandwidth but might reduce the number of states of the required gadget. Further, if one needs a UTM with some other special properties it is useful to know these too can be converted into gadgets.

▶ **Theorem 7.** *If there exists a universal Turing machine M with p symbols and q states then there exists a p state gadget G for which motion planning is PSPACE-complete with bandwidth $4q - 1$.*

Proof. We arrange the gadgets in the same way as Theorem 1. Each state of the gadget is a symbol of the tape. Each gadget has 4 locations for each TM state from combinations of in/out and left/right. Since these gadgets have $4q$ locations the furthest distance from one node in a gadget to another is $4q - 1$. ◀

4 Hardness for Reversible Deterministic Gadgets

In this section we briefly cover hardness for constant bandwidth reversible deterministic gadgets. We first note that a previous result from **Nondeterministic Constraint Logic (NCL)** [22] to locking 2-toggles only increases bandwidth by a constant factor. We then note that this implies hardness for the set of gadgets which simulating a locking 2-toggle.

4.1 Locking 2-Toggle

Hardness for constant bandwidth NCL was shown in [22]. We prove the reduction from NCL to Locking 2-Toggles presented in [17] preserves constant bandwidth. The Locking 2-Toggle is shown in Figure 1a.

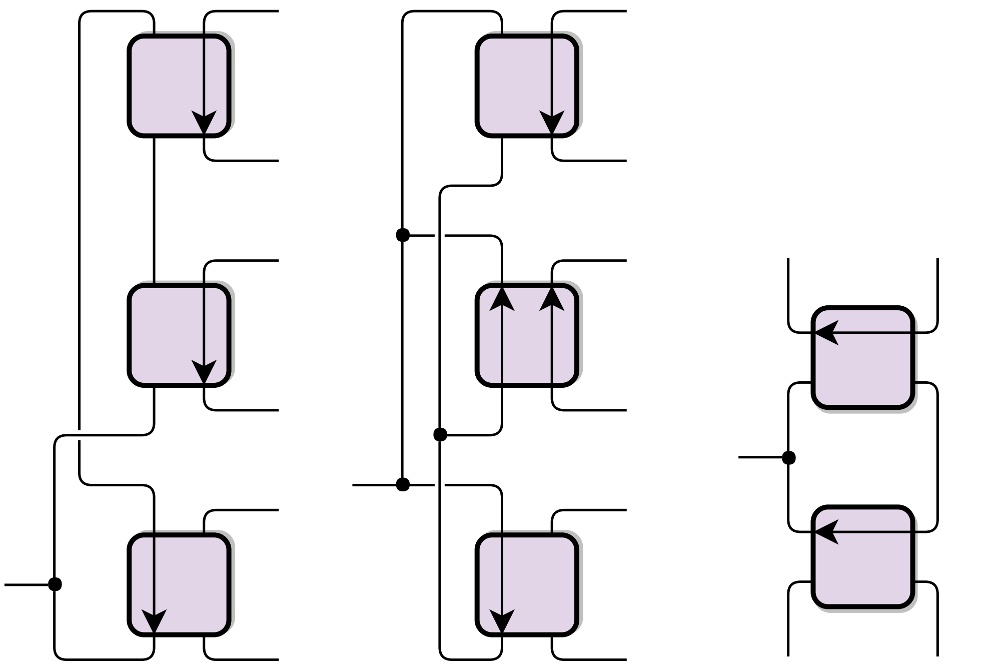
▶ **Theorem 8.** *Planar motion planning for the locking 2-toggle and branching hallway is PSPACE-complete even with $\mathcal{O}(1)$ bandwidth.*

Proof. Figures 4a and 4b shows NCL vertices from [17] each of which uses a constant number of motion planning gadgets. The right side of these gadgets connect to three edge gadgets from Figure 4c. The left-side node connects to a branching hallway which allows the agent to traverse to any of the AND and OR gadgets.

The edges on the right side of the gadgets retain constant distances, however the left side of the gadget requires more care based on how we define branches as a gadgets. If we do not define a branching gadget[2] and instead just have edges between locations this would have a single really long edge from the first gadget to the last. We instead model small branching gadgets which allow us to preserve the distance.

We arrange the gadgets as shown in Figure 5. We first place the branch for gadget which connects to the previous branch, next branch, and the left edge of its vertex. The vertex gadget is placed. If the vertex shares an edge with a vertex later in the sequence we place the edge gadget and connect it to the vertex gadget. If the edge connects to a previous vertex where we've already placed the edge gadget we connect it back. Each branch only connects to neighboring gadgets so these edges have constant length. The edges between edge gadgets are constant length because we can assume the NCL edges have constant length as well. ◀

[2] a single-state 3-location gadget with all allowed traversals



(a) NCL AND constructed from locking 2-toggle.

(b) NCL OR constructed from locking 2-toggle.

(c) NCL edge constructed from locking 2-toggle.

Figure 4 Constraint Logic Vertices and Edges constructed from Locking 2-Toggles.

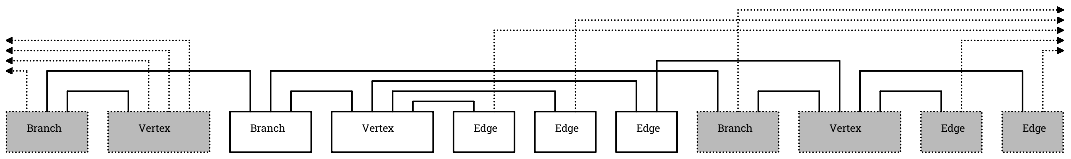
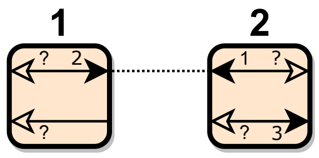


Figure 5 Arrangement of NCL gadgets and branches which preserves bandwidth. Each box is a gadget which is replaced by a gadget drawn above. Dotted lines indicate edges which go to gadgets not shown. Each edge is actually 2 motion planning edges. The left most gadget has edges which have all already appear and thus does not have any adjacent edge gadgets. The middle gadget has all edge on its right side including an edge shared with its neighbor.

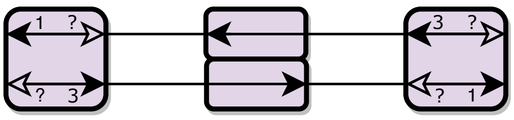
Results in [17] state that every interacting-k-tunnel reversible deterministic gadget simulates a locking 2-toggle. Since our definition of gadget bandwidth depends on gadget size we cannot achieve a constant bandwidth for this whole class of gadgets. However we show the hardness even the bandwidth is linear in gadget size.

▶ **Corollary 9.** *Motion planning for every interacting-k-tunnel reversible deterministic gadget is PSPACE-complete even with bandwidth $\mathcal{O}(k)$.*

Proof. The proof from [17] replaces each locking 2-toggle with the desired gadget. Due to the properties of the gadget we can make some assumptions about what traversals must exist in the gadget as in Figure 6a, and then use those traversals to simulate the locking 2-toggle. The simulation is shown in Figure 6b which also uses a 1 toggle which is a simple gadget that can be simulated. The edges still travel over the same number of gadgets, however the gadgets are now bigger by a factor of $2k$, since a k-tunnel gadget has $2k$ locations. ◀



(a) An arbitrary interacting-k-tunnel reversible deterministic gadget. Hollow arrows indicate traversals that may or may not be possible. Solid or absent arrows indicate traversals that are or are not possible, respectively.



(b) An arbitrary gadget and a 1-toggle simulate a locking 2-toggle.

Figure 6 Figures from [17] describing how interacting-k-tunnel reversible gadgets can simulate a locking 2-toggle.

5 Easiness for DAG gadgets

Interestingly, for DAG gadgets, we obtain a general FPT result for treewidth and number of states. Our algorithm uses dynamic programming over the possible traversals in each node of the tree decomposition. This results shows that building DAG gadgets is not sufficient for hardness. This also motivates further work into the example complexity of these subsets of gadgets.

▶ **Theorem 10.** *Motion planning for DAG gadgets of treewidth k and at most s states per gadget is FPT.*

Proof. First, we compute a tree decomposition $\mathcal{T}$ of the graph representation of width k in time $2^{\mathcal{O}(k^3)} \cdot p$ [14] for a factor p that is polynomial in the size of the given DAG gadgets. Without loss of generality, we assume that every node of $\mathcal{T}$ has at most 2 child nodes.

As data structure per tree decomposition bag, we keep track of different histories of locations (contained in the bag) and states, given in their relative order of traversal. Such a history is a sequence where each element (tuple) $\langle l, D\rangle$ comprises a location l and a corresponding state vector D indicating the state of all gadgets in the bag. Therefore there are up to $p = k \cdot s^k$ many tuples, resulting in up to $\mathcal{O}(p!)$ many sequences that are contained in a history. Indeed, while locations might reappear in a history multiple times, by the DAG property the state vector has to change and we can therefore bound their sizes.

Note that the graph representation ensures that we see all the gadget's locations in at least one (common) bag. In such a bag, we can easily check for a given history whether every location has exactly one successor and whether these transitions and state changes are actually sound, DAG, and reachable from its predecessor.

Initially, only the agent's start location and initial state vector configuration is reachable and for the bag that contains the goal location, we only store histories reaching the goal. Using the order among history sequences we only obtain paths reaching the goal. ◀

6 Hardness for Fixed State Changes

Here we show motion planning with a fixed number of state changes is W[SAT]-hard and in the class XP. We do so via reduction from ***k*-ones CNF SAT**. The number of tunnels of our gadget is linear in the number of appearances of a variable.

▶ **Theorem 11.** *Motion Planning for interacting-t-tunnel gadgets is W[SAT]-hard with respect to number of total state changes.*

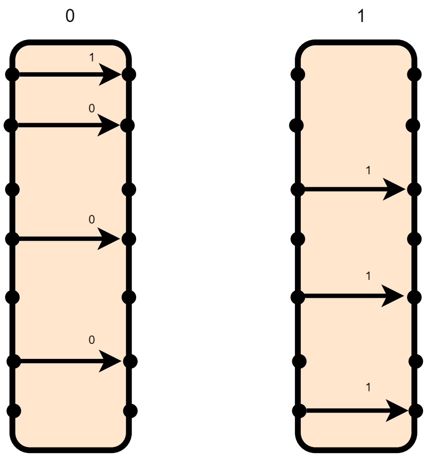


Figure 7 Variable Gadget for reduction where each variable appears in at most 3 clauses.

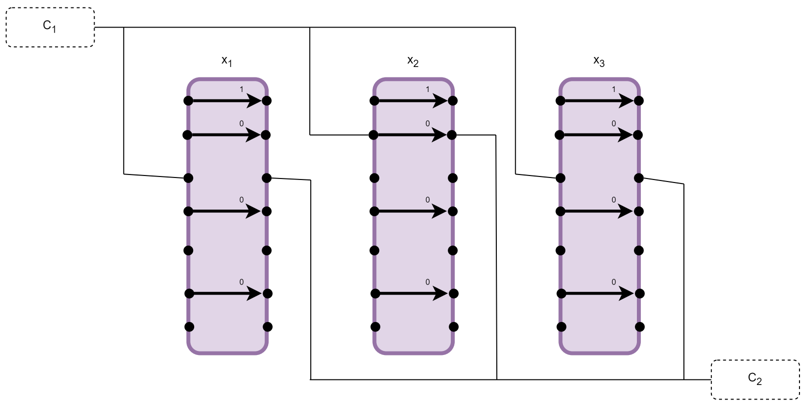


Figure 8 Example wiring for clause $c_1 = (x_1 \vee \overline{x_2} \vee x_3)$.

Proof. For each variable we create a $2t' + 1$ directed tunnel gadget (Figure 7) where t' is largest number of appearances of a variable. The gadget has two states, 0 and 1 corresponding to the assignment of the variable. At the top of the gadget we have a "assignment tunnel" which allows the gadget to go from state 0 to state 1. All other traversals do not change the state of the gadget. The other $2t$ tunnels are each paired as a false tunnel and a true tunnel, ensuring each variable gets its own pair of tunnels. In state 0 only the false tunnel are open and in 1 the true tunnels.

We connect the assignments tunnels all together to allow the agent to pick up to k gadgets to set to state 1. Then for each clause we have a location which are connected through tunnels of the literals that appear in the clause. This is shown in Figure 8. The agent must then go through true or false tunnel for some variable that satisfies the clause. The goal is to reach the location for the last clause.

If there exists a satisfying assignment with k ones then there exists a sequence of state changes to set up to k gadgets to state 1 then a path through all the true and false tunnels for each clause. If there exists a path to reach the target with only k state changes then there exists a satisfying assignment with k ones which satisfies each clause. ◀

It is important that the number of tunnels / locations is not fixed. If we reduce from k-ones SAT with a fixed variable appearance we should have fixed location gadgets. However this version of k-ones SAT is in FPT [18] with respect to fixed appearances, fixed arity of the clauses, and fixed k. Any hardness would require relaxations of these parameters or a different reduction. Our next results shows that this we cannot get paraNP-hardness for fixed state changes.

▶ **Theorem 12.** *Motion Planning for interacting-t-tunnel gadgets is in XP with respect to number of total state changes k.*

Proof. Let n be the number of gadgets in our system where each gadget has L locations. We then need to search over all length k sequences of traversals where we have up to nL^2 choices at each step. This requires searching through up to $(nL^2)^k$ possible choices. We may verify a sequence is correct since transitions which don't change the graph state can be verified in polynomial time. We only need to check the start of the next state changing traversal is reachable from the end of the previous traversal. ◀

7 Applications

Now we have all the results needed to prove consequences for programmable matter as well as video games and puzzles. We state two results involving motion planning for a model of DNA computing, uniform control robotics, and pivoting robots. We follow by showing that multiple video games which have been shown to be PSPACE-Complete with motion planning gadgets are now hard on constant height levels.

7.1 Programmable Matter

Here we highlight three results in the field of DNA computing and robot motion planning that are improved by our results. The first is in **Chemical Reaction Networks** on a Surface [20] which is a special case of asynchronous cellular automata. This model consists of a set of species or symbols, and a set of reactions of the format $A + B \to C + D$, which means when A is adjacent to B we may replace them with C and D respectively. The specific version of the model that was shown hard used only swap reactions of the form $A + B \to B + A$, thus the reactions only swap species. The second model is called **single step** [15], the case we consider has 1×1, 1×2 and 2×1 polyominoes on $n \times n$ board with no obstacles other than the edge of the board. The user gives a direction and all units on the board move uniformly in that direction unless blocks by the boundary or other units. The last model aims to characterize the **reconfiguration of pivoting robots** [1], these are hexagon or square tiles which live on the grid. These tiles must remain connected at all times and move by pivoting motions.

▶ **Theorem 13.** *The following parameterized problems are PSPACE-Complete*

- *1-reconfiguration in Swap Chemical Reaction Networks on a constant height level*
- *Tilt with dominoes and a constant height rectangle board*
- *Reconfiguration in modular pivoting robots with constant height starting and ending configuration*

Proof. These reductions all build constant sized gadgets. Most of these gadgets are a locking 2-toggle, or a gadget which can simulate a locking 2-toggle. We can arrange these gadgets all in the same row of the reduction instance and arrange the connections into wires above the gadgets similar to NCL in Figure 5. Then by assuming bandwidth k we can know that each connection between the gadgets only travels at most k distance away, which also means that only k wires can exist in the same column. Since each gadget is a constant size and there's only k wires at each column, the total height is $O(k)$.

Different programmable matter models have benefited from using gadgets. Reconfiguration in Surface Chemical Reaction Networks have been shown to be PSPACE-complete from [2] by building a locking 2-toggle. The wires are a single cell wide and can be drawn above as explained. thus we gain PSPACE-complete for a constant bandwidth surface. Locking 2-toggles with constant bandwidth is shown in Theorem 8.

The tilt model has tiles on a grid that move according to uniform global commands. Relocating a tile through one of these systems is PSPACE-complete with no obstacles but allowing dominoes [15] is hard with a constant height board using a crossing toggle lock which is a universal reversible deterministic gadget (Corollary 9). Finally in [1] reconfiguring modular pivoting robots on a square and hexagon grid is PSPACE-complete. This reduction reduces from "1-toggle-protected locking 2-toggle". This adds a 1-toggle to each edge. The reduction in the paper describes how to add 1-toggles to the reduction from NCL to locking 2-toggles. This reduction preserves bandwidth since each gadget only adds a constant number of 1-toggles. ◀

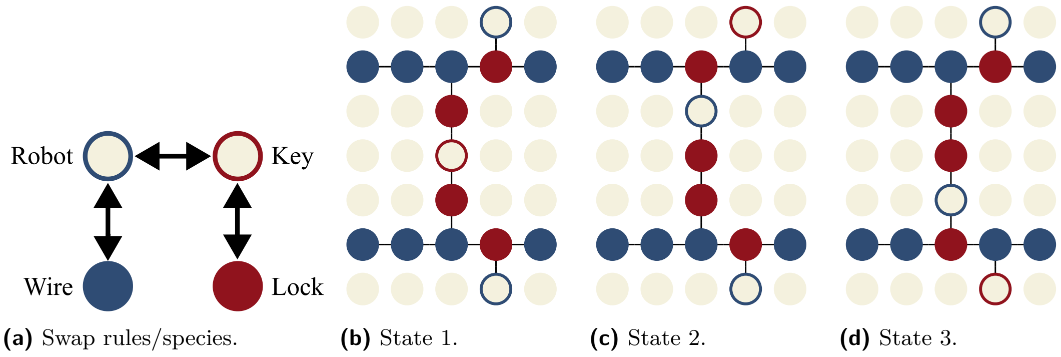


Figure 9 Locking 2-toggle implemented by swap rules. (a) The swap rules and species names. (b-d) The three states of the locking 2-toggle.

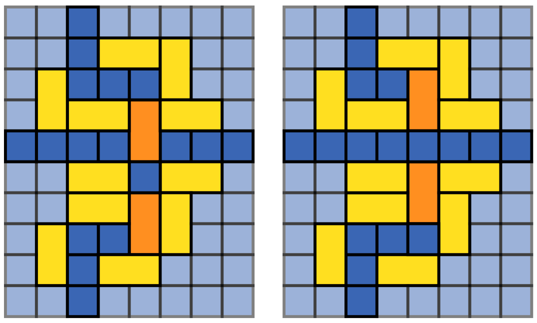


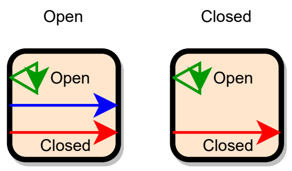
Figure 10 Crossing Toggle Lock from [15].

7.2 Games

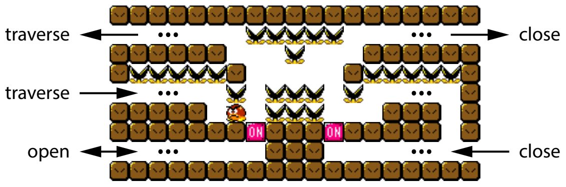
Many of these video games are described in [4]. We generalize the game by allowing larger maps and ignoring glitches. The decision problem we ask for these games is **Reachability**, given a starting location of the player and a target location, can the player reach the target.

Push and Pull puzzles are played on a grid with a single agent. The board also contains blocks which may be pushed or pulled. **PushPull-1F** allows the agent to push or pull a single block. The F means there exists fixed blocks which may not be pulled. **Sokobond** is a game where the player starts as a single atom. The board contains other atoms which are a single tile, and molecules which are larger. Both of these which may be pushed by the player. However each atom has a number of free electrons ≥ 0 and may merge with others which also have free electrons to form molecules.

Here we focus only on a couple games but this applies to many other variants. For example recent work has shown that many **Super Mario games** are hard [6], along with many games related to the Push and Pulling Puzzles [5].



(a) Directed open door gadget.



(b) Gadget in Super Mario.

Figure 11 Hardness for Super Mario by reducing from a variant of the directed open door gadget (a), which can be modeled as depicted in (b).

▶ **Theorem 14.** *The following parameterized games are PSPACE-Complete,*

- *Super Mario 64 with constant level height and depth*
- *Super Mario World with constant level height*
- *Donkey Kong Country 1, 2, and 3 with constant level height*
- *The Legend of Zelda: A Link to the Past with constant level height*
- *The Legend of Zelda: Oracle of Seasons with constant level height*
- *PushPull-1F on a constant height grid*
- *Sokobond on a constant height grid*

Proof. Similar to the Theorem 14 these games build constant sized gadgets and wires so the size of the configurations are within a constant factor of the bandwidth. Here many of the proofs are using door gadgets with some being reversible deterministic gadgets like the locking 2-toggle.

Various 2D and 3D Nintendo games have been shown to build universal gadgets which would then be PSPACE-complete for levels with only one dimension of non-constant size. For such as Super Mario 64 a symmetric self-closing door is created [8]. Each gadget is constant size in all 3 dimensions and wires are constant size. 13 different door gadget were given in [6], we give one here in Figure 11b showing hardness of Super Mario World. This uses a slight variation on the directed open required door where the input and output of the open tunnel are merged. Constant height levels of the 2D games The Legend of Zelda: A Link to the Past, and Donkey Kong Country 1, 2, and 3 [4] all use different variations of universal door gadgets (Theorem 2).

Sokobond [8] which use doors (Theorem 2) each with a larger molecules and the agent is a single H. The Legend of Zelda: Oracle of Seasons [16] which uses crossing two toggles which are reversible deterministic interacting tunnel gadgets (Theorem 9).

Famous puzzles have used results with gadgets such as pushing and pulling blocks puzzles like PushPull-1F [17] which builds a crossing locking 2-toggle which falls under Theorem 8. ◀

8 Conclusion and Future Work

We presented many classes of gadgets for which Motion Planning remains hard even on graphs of fixed bandwidth. Many models have problems which then inherit hardness from these results. However this is just an initial dive into parameterized complexity of gadgets and we leave some open problems.

- What is the exact constant needed for hardness of NCL and Locking 2-Toggles?
- Does there exist a gadget with hardness for bandwidth < 7? The set of graphs with bandwidth 1 is exactly the set of paths. This only includes systems of 1-Tunnel gadgets which are known to be easy [17].
- If we only fix the bandwidth (treewidth) and not the number of states we believe this problem is no longer in FPT. In what complexity class does this version of the problem lie?
- Are there any classes of gadgets more powerful than DAGs but still in NP that remain hard with respect to structural parameters? One potential class is LDAGs which allow for transitionless traversals.
- We believe that Fixed Global State Changes is in the class W[P] via reduction from Reachability to k-ones SAT on circuits and maybe even W[SAT]-Complete.
- We are unable to immediately reduce from our gadget in Theorem 11 to our applications such as Push-k with fixed number of block pushes. This is due to the fact the assignment tunnel transition affects the state of all other tunnels. This seems difficult to model as a cell in Push has fixed degree. Is there a small (constant location) gadget that can achieve this hardness as well?

References

1 Hugo A. Akitaya, Erik D. Demaine, Andrei Gonczi, Della H. Hendrickson, Adam Hesterberg, Matias Korman, Oliver Korten, Jayson Lynch, Irene Parada, and Vera Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *37th International Symposium on Computational Geometry (SoCG 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

2 Robert M. Alaniz, Josh Brunner, Michael Coulombe, Erik D. Demaine, Jenny Diomidova, Timothy Gomez, Elise Grizzell, Ryan Knobel, Jayson Lynch, Andrew Rodriguez, Robert Schweller, and Tim Wylie. Complexity of reconfiguration in surface chemical reaction networks. In Ho-Lin Chen and Constantine G. Evans, editors, *29th International Conference on DNA Computing and Molecular Programming (DNA 29)*, volume 276 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DNA.29.10`.

3 Robert M. Alaniz, Bin Fu, Timothy Gomez, Elise Grizzell, Andrew Rodriguez, Robert Schweller, and Tim Wylie. Reachability in restricted chemical reaction networks. *arXiv preprint arXiv:2211.12603*, 2022. `doi:10.48550/arXiv.2211.12603`.

4 Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015. `doi:10.1016/J.TCS.2015.02.037`.

5 Hayashi Ani, Josh Brunner, Erik D Demaine, Jenny Diomidova, Timothy Gomez, Della Hendrickson, Yael Kirkpatrick, Jeffery Li, Jayson Lynch, Ritam Nag, et al. Agent motion planning as block asynchronous cellular automata: Pushing, pulling, suplexing, and more. In *International Conference on Unconventional Computation and Natural Computation*, pages 219–236. Springer, 2024.

6 Hayashi Ani, Erik D Demaine, Holden Hall, Matias Korman, et al. Pspace-hard 2d super mario games: Thirteen doors. In *12th International Conference on Fun with Algorithms (FUN 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

7 Joshua Ani, Sualeh Asif, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, Jayson Lynch, Sarah Scheffler, and Adam Suhl. PSPACE-completeness of pulling blocks to reach a goal. *Journal of Information Processing*, 28:929–941, 2020. doi:10.2197/IPSJJIP.28.929.

8 Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yenhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Walking through doors is hard, even without staircases: Proving PSPACE-hardness via planar assemblies of door gadgets. In *10th International Conference on Fun with Algorithms (FUN 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

9 Joshua Ani, Lily Chung, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box Dude. In *11th International Conference on Fun with Algorithms (FUN 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

10 Joshua Ani, Michael Coulombe, Erik D. Demaine, Yevhenii Diomidov, Timothy Gomez, Dylan Hendrickson, and Jayson Lynch. Complexity of motion planning of arbitrarily many robots: Gadgets, Petri Nets, and Counter Machines. In *2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

11 Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Traversability, reconfiguration, and reachability in the gadget framework. *Algorithmica*, 85(11):3453–3486, 2023. doi:10.1007/S00453-023-01140-0.

12 Joshua Ani, Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. *Theoretical Computer Science*, page 113945, 2023. doi:10.1016/J.TCS.2023.113945.

13 Sanjeev Arora. *Computational Complexity: A Modern Approach.* Cambridge University Press, Cambridge, 1 edition, 2009.

14 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

15 David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie. Relocating units in robot swarms with uniform control signals is PSPACE-complete. In *Canadian Conference on Computational Geometry*, 2020.

16 Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *9th International Conference on Fun with Algorithms (FUN 2018)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

17 Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

18 Dániel Marx. Parameterized complexity of constraint satisfaction problems. *computational complexity*, 14:153–183, 2005. doi:10.1007/S00037-005-0195-9.

19 Turlough Neary and Damien Woods. Small weakly universal turing machines. In *International Symposium on Fundamentals of Computation Theory*, pages 262–273. Springer, 2009. doi:10.1007/978-3-642-03409-1_24.

20 Lulu Qian and Erik Winfree. Parallel and scalable computation and spatial dynamics with dna-based chemical reaction networks on a surface. In *International Workshop on DNA-Based Computers*, pages 114–131. Springer, 2014. doi:10.1007/978-3-319-11295-4_8.

21 Manuel Sorge and Mathias Weller. The graph parameter hierarchy, 2012–2020. URL: https://manyu.pro/assets/parameter-hierarchy.pdf.

22 Tom C. van der Zanden. Parameterized complexity of graph constraint logic. In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.