Trains, Games, and Complexity: 0/1/2-Player Motion Planning through Input/Output Gadgets

Joshua Ani[1], Erik D. Demaine[1][0000-0003-3803-5703], Dylan Hendrickson[1][0000-0002-9967-8799], and Jayson Lynch[2][0000-0003-0801-1671]

[1] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139, USA
[2] University of Waterloo, Waterloo, ON, Canada

Abstract. We analyze the computational complexity of motion planning through local "input/output" gadgets with separate entrances and exits, and a subset of allowed traversals from entrances to exits, each of which changes the state of the gadget and thereby the allowed traversals. We study such gadgets in the zero-, one-, and two-player settings, in particular extending past motion-planning-through-gadgets work [3,4] to zero-player games for the first time, by considering "branchless" connections between gadgets that route every gadget's exit to a unique gadget's entrance. Our complexity results include containment in L, NL, P, NP, and PSPACE; as well as hardness for NL, P, NP, and PSPACE. We apply these results to show PSPACE-completeness for certain mechanics in Factorio, [the Sequence], and a restricted version of Trainyard, improving the result of [1]. This work strengthens prior results on switching graphs, ARRIVAL [5], and reachability switching games [6].

Keywords: gadgets · motion planning · hardness of games

1 Introduction

Imagine a train proceeding along a track within a railroad network. Tracks are connected together by "switches": upon reaching one, the switch chooses the train's next track deterministically based on the state of the switch and where the train entered the switch; furthermore, the traversal changes the switch's state, affecting the next traversal. ARRIVAL [5] is one game of this type, where every switch has a single input and two outputs, and alternates between sending the train along the two outputs; the goal is to determine whether the train ever reaches a specified destination. Even this seemingly simple game has unknown complexity, but is known to be in NP $\cap$ coNP [5], so cannot be NP-hard unless NP = coNP. More recent work shows a stronger result of containment in UP $\cap$ coUP as well as CLS [9], PLS [11], and UEOPL [7]. But what about other types of switches?

In this paper, we introduce a very general notion of "input/output gadgets" that models the possible behaviors of a switch, and analyze the resulting complexity of motion planning/prediction (does the train reach a desired destination?) while navigating a network of switches/gadgets. This framework gives us an expressive set of problems with different complexity classes to use as the basis for reductions for other problems of interest. For example, it is related to the generalization of ARRIVAL in [6] which define Reachability Switching Games. The paper further also describes how these Reachability Switching Games are related to switching systems and Propp machines, both of independent interest. In addition to ARRIVAL, our model captures other toy-train models, including those in the video game Factorio or the puzzle game Trainyard. In some cases, we obtain PSPACE-hardness, enabling building of a (polynomial-space) computer out of a railway system with a single train. Intuitively, our model is similar to a circuit model of computation, but where the state is stored in the gates (gadgets) instead of the wires, and gates update only according to visits by a single deterministically controlled agent (the train).

This work builds off of prior work on the computational complexity of agent-based motion planning [3, 4], extending it zero-player situations. An analogous generalization of computational problems based on the number of players and boundedness of moves can be found in Constraint Logic [10] which has served as a basis for a large number of hardness proofs for reconfiguration problems, as well as games and puzzles. However this line of work differs from Constraint Logic because it involves the changes to the system being localized in a single agent, whereas all edges in a constraint logic puzzle are available for any given move. This is helpful in constructing hardness proofs where action is geographically constrained. Further Constraint Logic is an inherently reversible system and generalizing beyond that constraint can be helpful in hardness reductions.

Motion Planning through Gadgets. Our model is a natural zero-player adaptation of the ***motion-planning-through-gadgets*** framework developed in [4] (after its introduction at FUN 2018 [3]), so we begin with a summary of that framework. A ***gadget*** consists of a finite set L of ***locations*** (entrances/exits), a finite set S of ***states***, and for each state $s \in S$, a labeled directed graph $G_s = (L, E_s)$ on the locations, where a directed edge (a, b) with label s' means that an agent can ***traverse*** the gadget by entering the gadget at location a and exiting at location b while changing the state of the gadget to s'. In general, a location might serve as the entrance for one traversal and the exit for another traversal; however, we consider in this paper the special case where each location serves exclusively as an entrance or an exit, but not both. Equivalently, a gadget is specified by its ***transition graph***, a directed graph whose vertices are state/location pairs $\in S \times L$, where a directed edge from (s, a) to (s', b) represents that an agent can traverse the gadget from a to b if it is in state s, and that such traversal changes the gadget's state to s'. We sometimes also consider the ***state-transition graph*** of a gadget, which is the directed graph with a vertex for each state $\in S$ and a directed edge (s, s') for each transition from (s, a) to (s', b) for any $a, b \in L$.

A ***system of gadgets*** consists of a set of gadgets, their initial states, and a ***connection graph*** on the gadgets' locations. If two locations a, b of two gadgets (possibly the same gadget) are connected by a path in the connection graph, then an agent can traverse freely between a and b (outside the gadgets). (Equivalently, we can think of locations a and b as being identified.) Gadgets are ***local*** in the sense that traversing a gadget does not change the state of any other gadgets.

In ***one-player motion planning***, we are given the initial location of a single agent in a system of gadgets, and the problem asks whether there is a sequence of traversals that brings that agent to its goal location.

Past work [4] analyzed (and in many cases, characterized) the complexity of these motion-planning problems for gadgets satisfying a few additional properties, specifically, gadgets that are "reversible deterministic k-tunnel" or that are "DAG k-tunnel", defined as follows:

– A gadget is ***k-tunnel*** if it has $2k$ locations and there is a perfect matching, whose matching edges are called ***tunnels***, such that the gadget only allows traversals between endpoints of a tunnel.
– A gadget is ***deterministic*** if its transition graph has maximum out-degree ≤ 1, i.e., an agent entering the gadget at some location a in some state s can exit at only one location b and in only one state s'.
– A gadget is ***reversible*** if its transition graph has the reverse of every edge, i.e., every traversal could be immediately undone.
– A gadget is a ***DAG*** if it has an acyclic state-transition graph. Such gadgets can necessarily be traversed only a bounded number of times (at most the number of states).

Input/Output Gadgets and Zero-Player Motion Planning. We define a gadget to be ***input/output*** if its locations can be partitioned into ***input*** locations (entrances) and ***output*** locations (exits) such that every traversal brings an agent from an input location to an output location, and in every state, there is at least one traversal from each input location. In particular, deterministic input/output gadgets have exactly one traversal from each input location in each state. Note that input/output gadgets cannot be reversible nor DAGs, so prior characterizations [4] do not apply to this setting.

An input/output gadget is ***output-disjoint*** if, for each output location, all of the transitions to it (including those from different states) are from the same input location. This notion is still more general than k-tunnel: it allows a one-to-many relation from a single input to multiple outputs.

With deterministic input/output gadgets, we can define a natural ***zero-player motion-planning game*** as follows. A system of gadgets is ***branchless*** if each connected component of the connection graph contains at most one input location.[3] Intuitively, if an agent finds itself in such a connected component, then there is only one gadget location it can enter, uniquely defining how it should

[3] Originally in [3] the gadget model was inherently branchless and non-deterministic, 1-state 'branching hallway' gadgets were used to connect multiple locations.

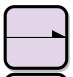
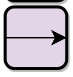
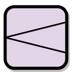
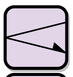
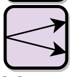
	**Set-Up Line**	A tunnel that can always be traversed in one direction and sets the state of the gadget to a specific state.
	**Toggle Line**	A tunnel that can always be traversed in one direction and toggles the state with each crossing.
	**Switch**	A three-location gadget with one input which transitions to one of two outputs depending on the state, without changing the state.
	**Set-Up Switch**	A switch which also sets the state of the gadget to a specific state.
	**Toggle Switch**	A switch which also toggles the state of the gadget with each crossing.

Table 1: Five subunits for 2-state, output-disjoint, input/output gadgets. We consider the 2-state gadgets to have the states *Up* and *Down*. Some subunits will set the state to a specific value such as Up, while some others always change the state when they are traversed.

proceed. (If an agent finds itself in a connected component with no input locations, it is stuck in a dead-end and the game ends.) We can think of edges in the connection graph as directed wires that point from output locations to the input location in the same connected component. Note branchless systems can still have multiple output locations in a connected component which functions as a fan-in.

In a branchless system of deterministic input/output gadgets, there are never any choices to make: in the connection graph, there is at most one reachable input location, and when the agent enters an input location there is exactly one transition it can make. Thus we define ***zero-player motion planning*** with a set of deterministic input/output gadgets to be the one-player motion-planning game restricted to branchless systems of gadgets. Lacking any agency, the decision problem is equivalent to whether the agent ever reaches the goal location while following the unique path available to it.

Classifying Output-Disjoint Deterministic 2-State Input/Output Gadgets. In this paper, we are primarily interested in output-disjoint deterministic 2-state input/output gadgets. In this section, we omit the adjectives and refer to them simply as "gadgets", and give a categorization of these gadgets, into 'trivial,' 'bounded,' and 'unbounded' gadgets. For each category, we will show that every gadget in the category can simulate at least one of a finite set of gadgets. The behavior of an input location to a gadget is described by how it changes the state and which output location it sends the agent to in each state. If the input location doesn't change the state and always uses the same output location, it can be ignored (the path can be 'shortcut' to skip that transition). Otherwise, the input location corresponds to one of the following five nontrivial subunits, and the gadget is a disjoint union of some of these subunits (which interact by sharing state). These subunits are given in Table 1.

The ARRIVAL problem [5] is equivalent to zero-player motion planning with the toggle switch: we replace each vertex in their switch graph with a toggle switch, or vice versa. We will use their terminology when referring to switch graphs in the ARRIVAL paper; however, when referring to gadgets in our model, a switch is a gadget (or part of a gadget) which does not change state when crossed. More generally, zero-player motion planning with an arbitrary set of deterministic single-input input/output gadgets (with gadgets specified as part of the instance) is equivalent to explicit zero-player reachability switching games, as defined in [6].

We call the states of any such two state gadget ***up*** and ***down***, and assume that each switch transitions to the top output in the up state and the bottom output in the down state; because we are not concerned with planarity, this assumption is fully general. There are two versions of the set line and set switch: one to set the gadget to each state. For example, a gadget with a set-up line and set-down switch is meaningfully different from a set-up line and set-up switch. We draw the set-down line and switch as the reflections of the set-up version. To represent the current state of a gadget, we make one of the lines in each switch dashed, so that the next transition would be made along a solid line. We categorize gadgets into three families:

1. ***Trivial*** gadgets have either no state change or no state-dependent behavior; they are composed entirely of either switches or toggle and set lines. They are equivalent to collections of simple tunnels, and zero-player motion planning with them is in L by straightforwardly simulating the agent for a number of steps equal to the number of locations.
2. ***Bounded*** gadgets have state-dependent behavior (i.e., some kind of switch) and one-way state change, either only to the up state or only to the down state. They naturally give rise to bounded games (a game in which the maximum number of turns is polynomially bounded before ending or repeating), because each gadget can change its state at most once.
3. ***Unbounded*** gadgets have state-dependent behavior and can change state in both directions. They naturally give rise to unbounded games.

We will find that the complexity of motion planning with a given gadget also depends on whether the gadget is ***single-input***, meaning it has only one input location, or multiple nontrivial inputs. A non-trivial input must contain at least one transition from it, and that transition must either change the state of the gadget or must not exist in all states of the gadget. The only nontrivial single-input gadgets are the set switch and the toggle switch, which are bounded and unbounded, respectively. Recall Table 1 gives definitions for the pieces of 2-state input-output gadgets. The full version of the paper proves for all 2-state input-output gadgets with multiple inputs, there is a system of those gadgets with the same behavior as one of eight gadgets made of pairs of the subunits.

Lemma 1. *Let G be an output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs.*

	Trivial (No state change or on tunnels)	**Bounded, multiple nontrivial inputs**	**Unbounded, multiple nontrivial inputs**
Zero-player (Fully Deterministic) [§2]	L	P-complete	PSPACE-complete
One-player [§??]	NL-complete	NP-complete	PSPACE-complete

Table 2: Summary of results for output-disjoint deterministic 2-state input/output gadgets.

	Contained in	**Hard for**
Zero-player (Fully Deterministic)	UP ∩ coUP [9]	NL (cf. [6])
One-player	NP (cf. [6])	NP (cf. [6])
Two-Player	EXPTIME (cf. [6])	PSPACE (cf. [6])

Table 3: Summary of results for single-input input/output gadgets. These results can be found in the full version of the paper [2].

– *If G is bounded, then it simulates either a switch/set-up line or a set-up switch/set-up line.*
– *If G is unbounded, then it simulates one of the following gadgets:*
  1. *switch/toggle line,*
  2. *switch/set-up line/set-down line,*
  3. *set-up switch/toggle line,*
  4. *set-up switch/set-down line,*
  5. *toggle switch/toggle line, or*
  6. *toggle switch/set-up line.*

Our Results. Table 2 summarizes our results on output-disjoint deterministic 2-state input/output gadgets. While our main motivation was to analyze zero-player motion planning, we also characterize the complexity of one-player motion planning for contrast. A full version of this paper is available [2].

We also consider motion planning with single-input input/output gadgets summarized in Table 3. This is a more immediate generalization of ARRIVAL [5], and is equivalent to the reachability switching games studied in [6]. We strengthen the results of [6] by showing that the containments in NP and EXPTIME still hold when we allow nondeterministic gadgets, and by showing hardness with specific gadgets—the toggle switch for zero-player, and each of the toggle switch and set switch for one- and two-player—instead of having gadgets specified as part of the instance.

In the full version of the paper, we apply this framework to prove PSPACE-completeness of the mechanics in several video games: one-train colorless Trainyard, the game [the Sequence], trains in Factorio, and transport belts in Factorio are all PSPACE-complete. The first result improves a previous PSPACE-completeness result for two-color Trainyard [1] by using a strict subset of game features. Factorio in general is trivially PSPACE-complete, as players have explicitly built computers using the circuit network; here we prove hardness for the restricted problems with only train-related objects and only transport-belt-related objects.

2 Zero Players

In this section, we consider unbounded gadgets with multiple inputs, which are naturally PSPACE-complete. The full version of the paper considers unbounded gadgets with only a single input and bounded gadgets with multiple inputs, which are naturally P-complete.

We show that zero-player motion planning with any unbounded output-disjoint deterministic 2-state input/output gadget which has multiple nontrivial inputs is PSPACE-complete through a reduction from Quantified Boolean Formula (QBF), which is PSPACE-complete, to zero-player motion planning with the switch/set-up line/set-down line, and by showing that every such gadget simulates the switch/set-up line/set-down line.

Edge Duplicators. Many of our simulations involve building an ***edge duplicator*** An edge duplicator is a construction which allows us to effectively make a copy of a line from X to X' in a gadget. For example, we might want to take a switch/toggle-line and build a three input gadget made of a switch and two separate toggle-lines. Edge duplication is achieved by routing two inputs A and B to X, and then sending the agent from X' to one of two exits A' or B' corresponding to the input used. The details of the construction of an edge duplicator depend on the gadget used; see Fig. 1 for an example.

2.1 PSPACE-hardness of the switch/set-up line/set-down line

In this section, we show that zero-player motion planning with the switch/set-up line/set-down line is PSPACE-hard through a reduction from QBF. The switch/set-up line/set-down line is a 2-state input/output gadget with three inputs: one sets the state to up, one sets it to down, and one sends the agent to one of two outputs based on the current state.

Theorem 2. *Zero-player motion planning with the switch/set-up line/set-down line is PSPACE-hard.*

A full proof can be found in the full version of the paper but a sketch is provided here. We first build an edge duplicator, shown in Fig. 1. This allows us to use gadgets with multiple set-up or set-down lines. Each quantifier gadget

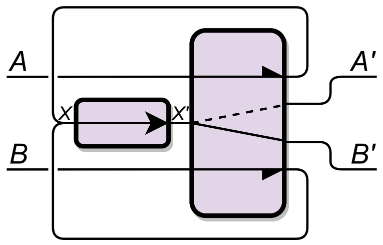


Fig. 1: An edge duplicator for the switch/set-up line/set-down line. A robot entering on the left sets the state of the switch, goes across the duplicated tunnel, and exits based on the state it set the switch to.

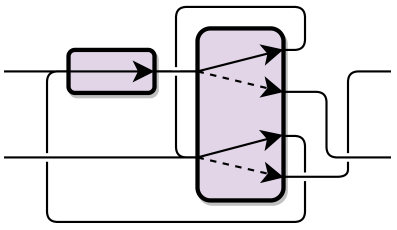


Fig. 2: An edge duplicator for the toggle switch/toggle switch. The tunnel on the left is duplicated.

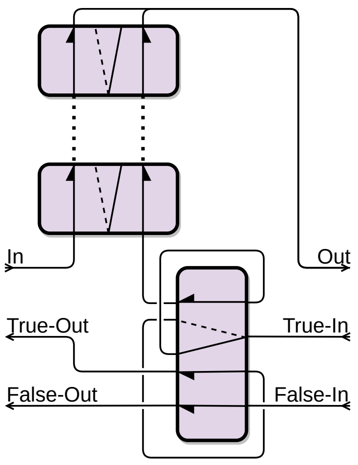


Fig. 3: The universal quantifier for the switch/set-up line/set-down line. An edge duplicator (Fig. 1) is used to give the bottom gadget two set-down lines.

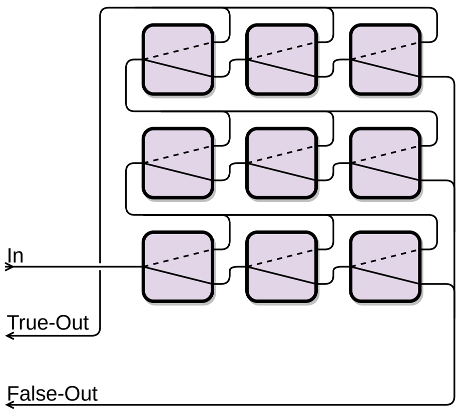


Fig. 4: Three clauses of CNF evaluation for the switch/set-up line/set-down line; each clause is a row of three switches. The switches are part of gadgets in the quantifiers. We assume the top exit of each switch corresponds to that literal being true.

has three inputs, called In, True-In, and False-In, and three outputs, called Out, True-Out, and False-Out. The agent will always first arrive at In. This sets the variable controlled by that quantifier to true, and the agent leaves at Out, which sends it to the next quantifier gadget. The universal quantifier gadget is shown in Fig. 3. The existential quantifier is identical except that True-Out and False-Out are swapped, and True-In and False-In are swapped.

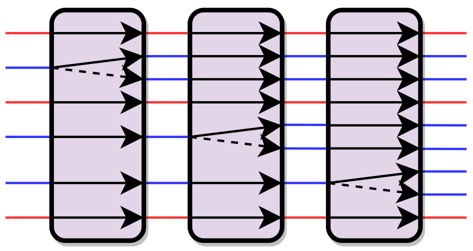


Fig. 5: A simulation of three toggle lines and three toggle switches from gadgets with one toggle switch and 5, 6, and 7 toggle lines. The red tunnels are toggle lines and the blue tunnels are toggle switches.

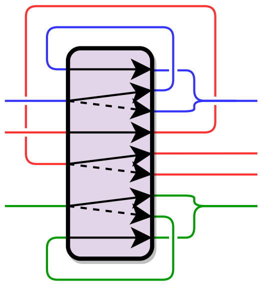


Fig. 6: A simulation of a switch/set-up line/set-down line from the gadget built in Fig. 5. The switch, set-up line, and set-down line are red, green, and blue, respectively.

2.2 Other gadgets simulate the switch/set-up line/set-down line

In this section, we show that every unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs can simulate the switch/set-up/set-down. We only need to show that the five other gadgets from Lemma 1 simulate the switch/set-up/set-down. It follows that zero-player motion planning with any such gadget is PSPACE-complete, since we can replace each gadget in a system of switch/set-up/set-down with a simulation of it. Some cases are presented here, see the full version of the paper for the remaining cases.

Toggle Switch/Toggle Switch. We begin with the toggle switch/toggle switch, which is not part of our basis of gadgets but will be a useful intermediate gadget. It builds an edge duplicator, shown in Fig. 2. We can merge the two outputs of one of the toggle switches to simulate a toggle switch/toggle line, and then duplicate the toggle line to make a gadget with one toggle switch and any number of toggle lines.By putting such gadgets in series, we can simulate a gadget with any number of toggle lines and any number of toggle switches. Fig. 5 shows this for three toggle lines and three toggle switches, which is as large as we need. This simulated gadget can finally simulate the switch/set-up line/set-down line, shown in Fig. 6.

Switch/Toggle Line. We first build an edge duplicator, shown in Fig. 7. We can then duplicate the toggle line and put one copy in series with the switch, constructing a toggle switch/toggle line.

Set-Up Switch/Toggle Line. We first build an edge duplicator, shown in Fig. 8. We then simulate the switch/toggle line, shown in Fig. 9.

Toggle Switch/Set-Up Line. We simulate a set-up line/set-down switch using the toggle switch/ set-up line, as shown in Fig. 10; this is equivalent to a set-up switch/set-down line.

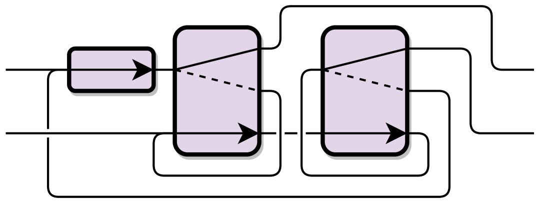


Fig. 7: An edge duplicator for the switch/toggle line. The leftmost tunnel is duplicated.

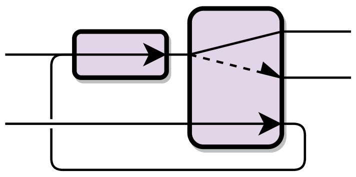


Fig. 8: An edge duplicator for the set-up switch/toggle line. The leftmost tunnel is duplicated.

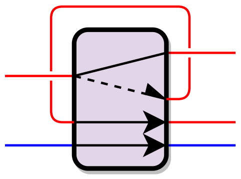


Fig. 9: A simulation of the switch/toggle line using the set-up switch/toggle line. Red is the switch and blue is the toggle line.

These simulations, together with Lemma 1, give the following theorem. The details of these cases are given in the full version of the paper.

Theorem 3. *Every unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs simulates the switch/set-up line/set-down line.*

Corollary 4. *Let G be an unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs. Zero-player motion planning with G is PSPACE-complete.*

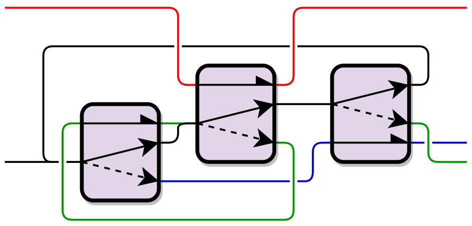


Fig. 10: A simulation of a set-up line/set-down switch from the set-up line/toggle switch. The state of the simulated gadget is the same as the state of the center gadget. The red path corresponds to the set-up line. When it enters the set-down switch, the robot goes along the blue lines if the state is down, the green lines if the state is up, and the black lines in both cases.

3 Open Problems

One interesting problem left open by our paper and several before it [5, 6, 9] is the complexity of zero-player motion planning with deterministic single-input input/output gadgets, or equivalently ARRIVAL and zero-player reachability switching games; this is known to be between NL-hard and NP $\cap$ coNP, which is a large gap. We conjecture that many of these single input gadgets are P-hard and we would be interested to see such a result. We also leave open the complexity of two-player one-agent motion planning, or two-player reachability switching games, which is between PSPACE-hard and EXPTIME.

Since input/output gadgets seem to be a natural and rich class of gadgets, one could expand our characterization of zero-player motion planning to include input/output gadgets beyond the output-disjoint deterministic 2-state ones. Another question is whether these gadgets remain hard in the planar case.

Finally, although we have only defined zero-player motion planning with input/output gadgets (and the Trainyard gadget), many other classes of gadgets could be explored in the zero-player model. This model begins to look a lot more like a typical circuit or computing model with the unusual constraint that only a single signal is ever propagating through the system. In particular, a reasonable zero-player motion planning problem with reversible, deterministic gadgets (like those studied in [3] and [4]) is similar to asynchronous ballistic reversible logic [8] introduced to explore potential low-power computing architectures.

Acknowledgments

We thank Jeffrey Bosboom for suggesting applying the gadget framework to railroad switches (specifically, a switch/tripwire gadget) in 2017, and Mikhail Rudoy for pointing us to the subsequent analysis of ARRIVAL [5]. We also thank Jeffrey Bosboom for providing simplified constructions for the set-up switch/set-down line and toggle switch/set-up line, and for general discussion on topics in and related to this paper. Some of this work was done during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) taught by Erik Demaine in Spring 2019. We thank the other participants of that class for related discussions and providing an inspiring atmosphere.

References

1. Almanza, M., Leucci, S., Panconesi, A.: Tracks from hell – when finding a proof may be easier than checking it. In: Ito, H., Leonardi, S., Pagli, L., Prencipe, G. (eds.) Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018). LIPIcs, vol. 100, pp. 4:1–4:13. La Maddalena, Italy (June 2018). https://doi.org/10.4230/LIPIcs.FUN.2018.4, https://doi.org/10.4230/LIPIcs.FUN.2018.4
2. Ani, J., Demaine, E.D., Hendrickson, D.H., Lynch, J.: Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. arXiv preprint arXiv:2005.03192 (2020)

3. Demaine, E.D., Grosof, I., Lynch, J., Rudoy, M.: Computational complexity of motion planning of a robot through simple gadgets. In: Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018). LIPIcs, vol. 100, pp. 18:1–18:21. La Maddalena, Italy (June 2018)
4. Demaine, E.D., Hendrickson, D., Lynch, J.: Toward a general theory of motion planning complexity: Characterizing which gadgets make games hard. In: Proceedings of the 11th Conference on Innovations in Theoretical Computer Science (ITCS 2020). pp. 62:1–62:42. Seattle, Washington (January 2020)
5. Dohrau, J., Gärtner, B., Kohler, M., Matoušek, J., Welzl, E.: Arrival: A zero-player graph game in NP $\cap$ coNP. In: A journey through discrete mathematics, pp. 367–374. Springer (2017)
6. Fearnley, J., Gairing, M., Mnich, M., Savani, R.: Reachability switching games. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). LIPIcs, vol. 107, pp. 124:1–124:14. Prague, Czech Republic (July 2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.124, https://doi.org/10.4230/LIPIcs.ICALP.2018.124
7. Fearnley, J., Gordon, S., Mehta, R., Savani, R.: Unique end of potential line. Journal of Computer and System Sciences **114**, 1–35 (2020)
8. Frank, M.P.: Asynchronous ballistic reversible computing. In: Proceedings of the IEEE International Conference on Rebooting Computing (ICRC). pp. 1–8. Washington, DC (November 2017)
9. Gärtner, B., Hansen, T.D., Hubáček, P., Král, K., Mosaad, H., Slívová, V.: ARRIVAL: next stop in CLS. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). LIPIcs, vol. 107, pp. 60:1–60:13. Prague, Czech Republic (July 2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.60, https://doi.org/10.4230/LIPIcs.ICALP.2018.60
10. Hearn, R.A., Demaine, E.D.: Games, puzzles, and computation. CRC Press (2009)
11. Karthik, C.: Did the train reach its destination: The complexity of finding a witness. Information Processing Letters **121**, 17–21 (2017)