# Traversability, Reconfiguration, and Reachability in the Gadget Framework

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

[1] Massachusetts Institute of Technology, Cambridge, MA, USA, {joshuaa,edemaine,diomidov,dylanhen}@mit.edu
[2] Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada, jayson.lynch@uwaterloo.ca

**Abstract.** Consider an agent traversing a graph of "gadgets", each with local state that changes with each traversal by the agent. Prior work has studied the computational complexity of deciding whether the agent can reach a target location given a graph containing many copies of a given type of gadget. This paper introduces new goals and studies examples where the computational complexity of these problems are the same or differ from the original relocation goal. For several classes of gadgets—DAG gadgets, one-state gadgets, and reversible deterministic gadgets—we give a partial characterization of their complexity when the goal is to traverse every gadget at least once. We also study the complexity of reconfiguration, where the goal is to bring the entire system of gadgets to a specified state. We give examples where reconfiguration is a strictly harder problem than relocating the agent, and also examples where relocation is strictly harder. We also give a partial characterization of the complexity of reconfiguration with reversible deterministic gadgets.

## 1 Introduction

The ***motion-planning-through-gadgets framework***, introduced in [3] and further developed in [4], captures a broad range of combinatorial motion-planning problems. It also serves as a powerful tool for proving hardness of games and puzzles that involve an agent moving in and interacting with an environment where the goal is to reach a specified location. Prior work [4] fully characterizes the complexity of 1-player motion planning with two natural classes of gadgets: ***DAG k-tunnel*** gadgets, which naturally lead to bounded games, and ***reversible deterministic k-tunnel*** gadgets, which naturally lead to unbounded games. Section 2 reviews these and other important definitions.

All of the prior work considers ***reachability***, where the decision problem is whether the agent can reach the target location.[3] In this paper, we begin extending the gadget model to victory conditions other than reaching a target

[3] Assembly and motion planning literature often use the term reachability to refer to whether an agent can reach a target location. However, reconfiguration literature

location. In particular we examine the complexity of reconfiguring a system of gadgets and of visiting every single gadget. These extensions seem natural and interesting, but are also motivated by the fact that this model has been used to show hardness of reconfiguration problems and problems with Hamiltonian Path like constraints.

We consider the ***universal traversal*** problem of whether the agent can visit every gadget. In Section 3, we characterize the complexity of this problem for three classes of $k$-tunnel gadgets: DAG gadgets, one-state gadgets, and reversible deterministic gadgets. Of particular note is that universal traversal can be harder than reachability for the same gadget. In particular, there are DAG $k$-tunnel gadgets for which reachability is in P but universal traversal is NP-complete. Additionally, reachability for one-state gadgets is always in NL, but universal traversal can be NP-complete.

In Section 4 we consider the ***reconfiguration*** problem of whether the agent can cause the entire system of gadgets to reach a target configuration. We exhibit a gadget with non-interacting tunnels for which reconfiguration is PSPACE-complete, but reachability is in P. We also show that for reversible gadgets, reconfiguration is at least as hard as reachability. In contrast, we exhibit a nonreversible gadget for which the reconfiguration is contained in P while reachability is NP-complete. The gadgets framework has already been used to prove complexity results about reconfiguration problems related to swarm [2] and modular robotics [1], so understanding reconfiguration in the gadgets model may provide an easier and more powerful base for such applications.

## 2 Gadget Model

We now define the gadget model of motion planning, introduced in [3].

A ***gadget*** consists of a finite number of ***locations*** (entrances/exits) and a finite number of ***states***. Each state $S$ of the gadget defines a labeled directed graph on the locations, where a directed edge $(a, b)$ with label $S'$ means that an agent can enter the gadget at location $a$ and exit at location $b$, changing the state of the gadget from $S$ to $S'$. Each of these arcs is called a ***transition***. Sometimes we will discuss a ***traversal*** from some location $a$ to location $b$ which refers to any possible transition from $a$ to $b$ in state $s$. Different states in a gadget can have different transitions while having the same traversability, because the transitions in those different states go from the same entrances to the same exits. Equivalently, a gadget is specified by its ***transition graph***, a directed graph whose vertices are state/location pairs, where a directed edge from $(S, a)$ to $(S', b)$ represents that the agent can traverse the gadget from $a$ to $b$ if it is in state $S$, and that such traversal will change the gadget's state to $S'$. Gadgets are ***local*** in the sense that traversing a gadget does not change the state of any other gadgets. An example can be seen in Fig. 1.

uses the term to refer to whether a target location in the configuration space is reachable from another. This would be equivalent to our reconfiguration problem which also specifies a target location for the agent.

A ***system of gadgets*** consists of gadgets, the initial state of each gadget, and an undirected ***connection graph*** on the gadgets' locations. If two locations $a$ and $b$ 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 $b$ along the connection graph. The ***configuration*** of a system of gadgets is that system of gadgets along with a state for each of the gadgets in the system. (Equivalently, we can think of locations $a$ and $b$ as being identified, effectively contracting connected components of the connection graph.) These are all the ways that the agent can move: exterior to gadgets using the connection graph, and traversing gadgets according to their current states. An agent's ***path*** is a sequence of valid transitions through gadgets and moves in the connection graph.

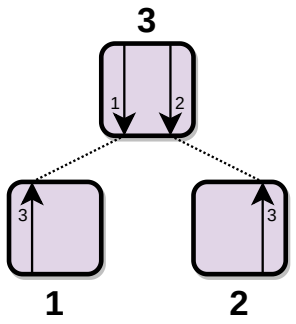



Fig. 1: A diagram describing the locking 2-toggle gadget. Each box represents the gadget in a different state, in this case labeled with the numbers $1, 2, 3$. Arrows represent transitions in the gadget and are labeled with the states to which those transition take the gadget. In the top state 3, the agent can traverse either tunnel going down, which blocks off the other tunnel until the agent reverses that traversal. Dotted lines help visualize the associated transitions between states.

**Definition 1.** *For a finite set of gadgets $F$, **reachability for $F$** is the following decision problem. Given a system of gadgets consisting of $n$ copies of gadgets in $F$, and a starting location and a win location in that system of gadgets, is there a path the agent can take from the starting location to the win location?*

We will consider several specific classes of gadgets.

A ***$k$-tunnel*** gadget has $2k$ locations, which are partitioned into $k$ pairs called ***tunnels***, such that every transition is between two locations in the same tunnel.

The ***state-transition graph*** of a gadget is the directed graph which has a vertex for each state, and an edge $S \to S'$ for each transition from state $S$ to $S'$. A ***DAG*** gadget is a gadget whose state-transition graph is acyclic. DAG gadgets naturally lead to problems with a polynomially bounded number of transitions, since each gadget can be traversed a bounded number of times. The complexity of the reachability problem for DAG $k$-tunnel gadgets, as well as the 2-player and team games, is characterized in [4].

A gadget is ***deterministic*** if every traversal goes to only one state and every location has at most 1 traversal from it. More precisely, its transition graph has maximum out-degree 1.

A gadget is ***reversible*** if every transition can be reversed. More precisely, its transition graph is undirected.

Reversible deterministic gadgets are gadgets whose transition graphs are partial matchings, and they naturally lead to unbounded problems. Prior work [4]

characterizes the complexity of reachability for reversible deterministic $k$-tunnel gadgets and partially characterizes the complexity of 2-player and team games.

A $k$-tunnel gadget has a ***distant opening*** if there is a transition in some state across a tunnel which opens a different tunnel. A tunnel is ***opened*** if a transition has taken it from a state where the tunnel did not have traversability in some direction to a state where it is now traversable.

In Section 3.2, we consider ***one-state***, $k$-tunnel gadgets. A transition in a gadget with only one state does not change the state, so the legal traversals never change.

## 3 Universal Traversal

In this section, we consider whether an agent in a system of gadgets can make a traversal across every gadget, called the ***universal traversal*** problem.

**Definition 2.** *For a finite set of gadgets $F$,* ***universal traversal for F*** *is the following decision problem. Given a system of gadgets consisting of $n$ copies of gadgets in $F$, and a starting location and a win location in that system of gadgets is there a path the agent can take from the starting location which makes at least one traversal in every gadget?*

We provide a full characterization for the complexity of this problem for three classes of gadgets. In Section 3.1, we characterize DAG $k$-tunnel gadgets. Universal traversal is NP-hard for some DAG gadgets where reachability is in P. This is somewhat similar to the distinction between finding paths and finding Hamiltonian paths. In Section 3.2, we further emphasize this difference by characterizing one-state $k$-tunnel gadgets. Reachability is always in NL for one-state gadgets, but we find that universal traversal is often NP-complete. Finally, in the full version of the paper we consider the unbounded case by characterizing universal traversal for reversible deterministic $k$-tunnel gadgets. In this case, the dichotomy is the same as for reachability.

### 3.1 DAG Gadgets

In this subsection, we consider universal traversal for $k$-tunnel DAG gadgets and show this problem is NP-hard for any DAG gadget which has and actually uses at least 2 tunnels, in the sense defined below. For some simple 1-tunnel DAG gadgets, universal traversal is analogous to finding Eulerian paths and is thus in P; however, more complex 1-tunnel gadgets can not easily be converted to an Eulerian path problem. For example the 1-toggle which switches direction after each transition or a gadget which can be traversed at most twice. We leave the case of 1-tunnel DAG gadgets open.

**Open Problem 1** *Is universal traversal with any 1-tunnel DAG gadget in P? Are there 1-tunnel DAG gadgets for which universal traversal is NP-complete?*

Some $k$-tunnel DAG gadgets with $k > 1$ act like 1-tunnel gadgets in that it is never possible to make use of multiple tunnels. A simple example is shown in Fig. 2. We formalize this notion in the following definition.

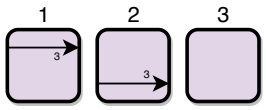



Fig. 2: A 2-tunnel DAG gadget which is not true 2-tunnel.

**Definition 3.** *A state of a $k$-tunnel gadget is **true 2-tunnel** if there are at least two tunnels, each of which is traversable in some state reachable (through any number of transitions) from that state. A gadget is **true 2-tunnel** if it is a $k$-tunnel gadget and has a true 2-tunnel state.*

Note that a $k$-tunnel gadget does not need multiple tunnels traversable in the same state to be true 2-tunnel: perhaps traversing the single traversable tunnel opens another tunnel. To justify this definition, we prove the following result.

**Theorem 1.** *Let $G$ be a $k$-tunnel which is not true 2-tunnel. Then there is a 1-tunnel gadget $G'$ and a bijection between states of $G$ to states of $G'$ such that replacing each copy of $G$ in a system of gadgets with a copy of $G'$ in the corresponding state gives an equivalent system of gadgets with respect to reachability and universal traversal.*

We will use the fact that every nontrivial DAG gadget simulates either a directed or an undirected single-use path, since we can take a final nontrivial state of the gadget [4]. The rest of this subsection is devoted to proving NP-completeness for universal traversal for true 2-tunnel DAG gadgets.

**Theorem 2.** *Universal traversal with any true 2-tunnel DAG gadget is NP-complete.*

To prove Theorem 2, we will focus on a ***final*** true 2-tunnel state of a DAG gadget, and only use the two tunnels which make this state true 2-tunnel. A final true 2-tunnel state is a true 2-tunnel state from which no other true 2-tunnel state can be reached. Such a state exists because the state-graph is a DAG. After making a traversal in this state, any resulting state is not true 2-tunnel, so only one of the two tunnels can be traversed in the future. If the gadget is nondeterministic, the agent may be able to choose which of the two tunnels this is. We consider several cases for the form of the last true 2-tunnel state, and show NP-hardness for each one. Most proofs are left to the full version of the paper.

The first case we consider is when the final true 2-tunnel state being considered has a distant opening.

**Lemma 1.** *Let $G$ be a true 2-tunnel gadget and let $S$ be a final true 2-tunnel state of $G$. If there exists a transition from $S$ across one tunnel which opens a traversal across another tunnel, then universal traversal for $G$ is NP-hard.*

*Proof.* We will only use the two tunnels involved in the opening transition from $S$ to $S'$ where $S'$ has some traversal which was not possible in $S$. Suppose traversing the top tunnel from left to right allows the agent to open the left-to-right traversal on the bottom tunnel. Then state $S$ has one of the two forms shown in Fig. 3, depending on whether the bottom tunnel can be traversed right to left in $S$. In either case, the top tunnel may or may not be traversable from right to left in $S$. Since $S$ is a final true 2-tunnel state, only the bottom tunnel is traversable in $S'$.

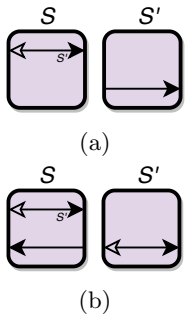



Fig. 3: Two cases for the form of the gadget in Lemma 1, assuming traversing the top tunnel to the right opens the bottom tunnel to the right. In (a) the bottom tunnel is not traversable to the left in state $S$ and in (b) it is. Unfilled arrows are traversals that may or may not exist depending on the gadget. Unlabled transitions may be to arbitrary states not specified here.

To show NP-hardness of universal traversal with true 2-tunnel gadget $G$, we reduce from reachability for $G$. Since the gadget has a distant opening, reachability is NP-complete [4]. We modify the system of gadgets in an instance of the reachability problem by adding a construction to each gadget which allows the agent to go back and make a traversal in it after reaching the win location. If the agent can reach the win location, it can then use any gadgets it did not already use, and if it cannot reach the win location, it cannot use the gadgets in this construction.

The construction is slightly different depending on whether the bottom tunnel can be traversed from right to left in state $S$. We use the construction in either Fig. 4 or Fig. 5. In either case, the agent cannot use the newly added gadgets until it first reaches the win location. Once it reaches the win location, it can

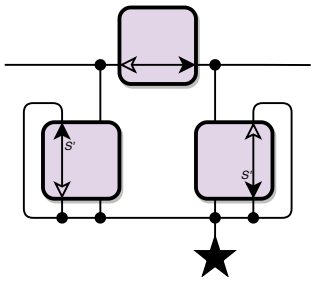



Fig. 4: The construction to allow the agent to use a gadget after reaching the win location (the star), when the bottom tunnel isn't traversable in state $S$ (the case of Fig. 3a).

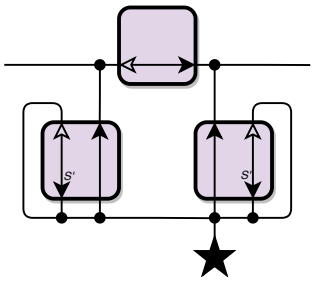



Fig. 5: The construction to allow the agent to use a gadget after reaching the win location (the star), when the bottom tunnel is traversable from right to left in state $S$ (the case of Fig. 3b).

open tunnels in the added gadgets, traverse the (top) gadget the construction is attached to, and return. If the agent already used the gadget this is attached to, it can instead use a traversal in each added gadget without visiting that gadget. So it is possible to make a traversal in every gadget if and only if the original reachability problem is solvable. □

Now we will assume the final true 2-tunnel state has no distant opening. If only one tunnel is traversable in this state, then it cannot be true 2-tunnel because the other tunnel will never become traversable. So both tunnels are traversable, and after making any traversal, there is only one tunnel which will ever be traversable. With no distant opening, we first consider the case where at least one of the tunnels is directed in the final true 2-tunnel state.

**Lemma 2.** *Let $G$ be a true 2-tunnel gadget and let $S$ be a final true 2-tunnel state of $G$. Suppose no transition from $S$ across one tunnel opens a traversal across the other tunnel. If, in $S$, some tunnel can be traversed in one direction but not in the other, then universal traversal for $G$ is NP-hard.*

The remaining case is when, in the final true 2-tunnel state, there is no distant opening and all tunnels are undirected. We branch into two cases one last time, based on whether traversing one tunnel requires closing the other tunnel. These can be found in the full version of the paper.

**Open Problem 2** *Is universal traversal restricted to planar systems of gadgets NP-hard for all true 2-tunnel DAG gadgets?*

### 3.2 One-State Gadgets

In this subsection, we consider universal traversal for $k$-tunnel gadgets with only one state. The reachability problem is clearly in NL for such gadgets, but we will see that universal traversal is often NP-complete.

A one-state $k$-tunnel gadget consists of directed and undirected tunnels, and is determined by the number of each type; we assume there is no untraversable tunnel since such a tunnel can be removed without affecting the problem. We fully characterize the complexity of universal traversal for such gadgets. We only prove a key lemma here, the rest can be found in the full version of the paper.

**Theorem 3.** *Let $G$ be a one-state $k$-tunnel gadget. If $G$ has no directed tunnels, then universal traversal for $G$ is in L. Otherwise, if $k \leq 2$ universal traversal for $G$ is NL-complete and if $k \geq 3$ universal traversal for $G$ is NP-complete.*

**Lemma 3.** *Universal traversal with any one-state $k$-tunnel gadget is in NL if $k \leq 2$.*

*Proof.* We provide an algorithm which runs in nondeterministic logarithmic space with an oracle for reachability in directed graphs. This shows that the universal traversal problem is in $\mathrm{NL}^{\mathrm{NL}}$. The algorithm can be adapted to run in NL by first using the oracle to convert the problem to an instance of 2SAT. It then solves this instance, since 2SAT is in NL.

The 2SAT formula has a variable for each tunnel in the system of gadgets; a satisfying assignment will provide a set of tunnels we can traverse to solve the universal traversal problem. For each gadget with tunnels $x_1$ and $x_2$, we have a clause $x_1 \vee x_2$ (if the gadget has only one tunnel, $x_1 = x_2$). For each pair of distinct tunnels $x$ and $y$, we query the reachability oracle to determine whether there is a path from the exit of $x$ to the entrance of $y$ or from the exit of $y$ to the entrance of $x$ (if $x$ or $y$ is undirected, we can use either location as the entrance or exit). If there is no path in either direction, we have a clause $\neg x \vee \neg y$.

We prove that this algorithm works, and then adapt it to an $\mathrm{L}^{\mathrm{NL}}$ algorithm which is known to equal NL [6]. Suppose the universal traversal problem is solvable, and consider the assignment which contains the tunnels which are used in the solution. Since the solution must use a tunnel in every gadget, each clause $x_1 \vee x_2$ is satisfied. If the solution uses both tunnels $x$ and $y$, there must be a path in some direction between $x$ and $y$, namely the path the agent takes between the two tunnels. For each clause $\neg x \vee \neg y$ in the formula, there is no such path, so the solution does not use both tunnels $x$ and $y$, so the clause is satisfied.

Now suppose the 2SAT formula is satisfiable, and consider the set $T$ of tunnels corresponding to true variables in a satisfying assignment. Because of the clauses $x_1 \vee x_2$, $T$ must contain a tunnel in each gadget. We define a relation $\to$ on $T$ where $x \to y$ if there is a path from the exit of $x$ to the entrance of $y$. As suggested by the notation, this relation is transitive: if $x \to y \to z$, there is a path from the exit of $x$ to the entrance of $y$, across $y$, and then to the entrance of $z$, so $x \to z$. Since each clause $\neg x \vee \neg y$ is satisfied, for any distinct $x, y \in T$ we have $x \to y$ or $y \to x$. That is, $\to$ is a strict total pre-order.

Then there must be a (strict) total order $\prec$ on $T$ such that $x \prec y \implies x \to y$: define another relation $\sim$ where $x \sim y$ if $x = y$ or both $x \to y$ and $y \to x$. Then $\sim$ is clearly an equivalence relation, and $\to$ is a total order on $T/\sim$. We can construct $\prec$ by putting the equivalence classes under $\sim$ in order according to $\to$, and arbitrarily ordering the elements of each equivalence class.

This now shows that there exists a set of locations from which a universal traversal is possible. The last step is to nondeterministically check that the start location of the agent has no strict predecessor in the preorder. This can be done by checking that the starting location is in the same equivalence class as the minimal element in our chosen total ordering. The agent can traverse the tunnels in $T$ in the order described by $\prec$. This is a solution to the universal traversal problem.

We run the algorithm in nondeterministic logarithmic space as follows. Begin with an NL algorithm that solves 2SAT, and assume the input is given in a format where we can check whether a clause $a \vee b$ is in the formula by checking a single bit for literals $a$ and $b$. For example, the input can be given as a matrix with a row and column for each literal. We run this nondeterministic 2SAT algorithm, except that whenever we would read a bit of the input, we perform a procedure to determine whether that clause is in the formula.

Suppose the algorithm to solve universal traversal wants to know whether $a \vee b$ is in the formula. If $a$ and $b$ are both positive literals, we simply check

whether they correspond to tunnels in the same gadget. If $a$ and $b$ have different signs, the clause is not in the formula. The interesting case is when $a = \neg x$ and $b = \neg y$ for tunnels $x$ and $y$, where we need to determine whether there is a path from the exit of $x$ to the entrance of $y$ or vice-versa.

In this case, we nondeterministically guess whether the clause exists, and then check whether the guess was correct. If we guess it does exist, we run a coNL algorithm to verify that there is no path from the exit of $x$ to the entrance of $y$ or vice versa; this can be converted to an NL algorithm. If the verification succeeds, we proceed; if it fails, we halt and reject. Similarly, if we guess the clause does not exist, we run an NL algorithm to verify that there is such a path, proceeding on success and rejecting on failure.

Consider the computation branches which have not rejected after this process. If the clause exists, the branch which attempted to verify it does not exist has entirely rejected, and the branch which attempted to verify it does exist has succeeded in at least one branch. So there is at least one continuing branch, and every such branch believes that the clause exists. Similarly if the clause does not exist, we end up with only branches which guessed that it does not exist. □

## 4 Gadget Reconfiguration

In this section we study the question of whether an agent has a series of moves after which the system of gadgets will be in some target configuration. In Section 4.1 we show that for reversible deterministic gadgets the reconfiguration problem is always PSPACE-complete if the reachability problem is PSPACE-complete. Section 4.2 shows some methods for constructing new PSPACE-complete gadgets from known ones and shows the reconfiguration problem can be PSPACE-complete even when a gadget does not change traversability. Finally, in Section 4.3, we show an interesting connection between reconfiguration problems and bounded reachability problems, expanding the classes of gadgets known to be in NP. We also exhibit, a gadget for which the reachability question is NP-complete but the reconfiguration question is in P.

**Definition 4.** *For a finite set of gadgets $F$, **reconfiguration for $F$** is the following decision problem. Given a system of gadgets consisting of $n$ copies of gadgets in $F$, a target configuration for that system of gadgets, and a starting location is there a path the agent can take from the starting location which makes the configuration of the system of gadgets equal to the target configuration?*

### 4.1 Reconfiguring Reversible Gadgets

**Theorem 4.** *For any set of reversible gadgets containing at least one gadget which is able to change state, there is a polynomial-time reduction from reachability with those gadgets to reconfiguration with those gadgets.*

*Proof.* We use the same technique as used to show that reconfiguration Nondeterministic Constraint Logic is PSPACE-complete [5]. We are given an instance

of the reachability problem, which is a network of gadgets with a target location. At the target location, add a loop with a single gadget which permits a traversal which changes its state. For the reconfiguration problem, we set the target states of all but the newly added gadget to be the same as the initial states, and we set the target state of the added gadget to be one reachable by making a traversal in the loop. If the reachability problem is solvable, the reconfiguration problem can be solved by navigating to the target location, traversing the loop through the added gadget, and taking the inverse transitions of the path taken to the target location to restore all other gadgets to the initial state. If the reconfiguration problem is solvable, its solution must involve visiting the added gadget, so the reachability problem is solvable. □

### 4.2 Verified Gadgets and Shadow Gadgets

In this section we will discuss a technique for generating gadgets for which the reconfiguration and reachability problems are computationally hard. The main idea is constructing a gadget which behaves well when used like a gadget with a known hardness reduction, but might also have other transitions which are allowed but put the gadget into some undesirable state.

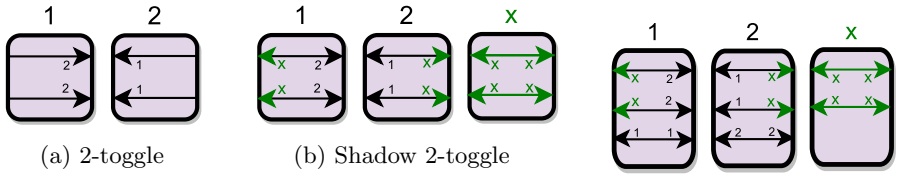
First, we will pick some base gadget which we want to modify. Next we will add additional ***shadow states*** to the gadget and additional transitions with the restriction that all newly added transitions must take the gadget to a shadow state. We call such a construction a ***shadow gadget*** of the base gadget. This has the nice property that if the agent takes any transition not be allowed in the base gadget, then the gadget will always stay in a shadow state after that transition.

**Theorem 5.** *Reconfiguration with a shadow gadget is at least as hard as reconfiguration with the base gadget.*

**Corollary 1.** *There is a gadget which never changes its traversability but with which reconfiguration is PSPACE-complete.*

Fig. 6 contains a diagram of the 2-toggle which is PSPACE-complete for reachability [3] and an example of a shadow 2-toggle which is PSPACE-complete for reconfiguration and is a gadget which never changes traversability (if there is a transition from some location $a$ to another location $b$ in any state, there must be a transition from $a$ to $b$ in every state). In fact all tunnels are always traversable in both directions. Finally, the figure shows a verified 2-toggle which is PSPACE-complete for reachability and a construction we will discuss next.

A ***verified gadget*** is a shadow gadget with some additional structure. From a shadow gadget we add two more locations, the ***verifying locations*** to the gadget. We may also add ***verified states*** which can only be reached by transitions from the added locations while the gadget is in normal states. We now add transitions among the verifying locations such that these locations can be connected in a series, there is always a traversal from the first to the last location if the gadget is in a normal state, and there is no such traversal if the gadget is in a shadow state. We call this added traversal the ***verification traversal***.




(a) 2-toggle (b) Shadow 2-toggle (c) Verified 2-toggle

Fig. 6: Examples of a PSPACE-complete gadget and constructions of a shadow gadget and verified gadget based on it.

**Theorem 6.** *Reachability with a verified gadget is at least as hard as reachability with the base gadget.*

### 4.3 Reconfiguration and DAG-like Gadgets

In [4] we study DAG gadgets as a naturally bounded class of gadgets. We now consider a generalized class of gadgets and describe cases in which the reachability question remains in NP.

For a finite family of gadgets $F$, we call a gadget ***F-DAG-like*** if its state graph can be decomposed into disjoint subgraphs for which those subgraphs are gadgets in $F$ and the transitions between these subgraphs are acyclic. We call the transitions between the subgraphs ***F-DAG-like transitions***.

With this notion, one may wonder what gadgets can be used in an $F$-DAG-like gadget and have the resulting reconfiguration or rechability problem with that gadget gadget still be in NP. We then show that if $F$ is a family of gadgets for which the reconfiguration problem is in NP, then the reconfiguration and reachability problems for $F$ and for $F$-DAG-like gadgets are also in NP. We call a finite set of gadgets ***NPReDAG*** if they are all $F$-DAG-like for some fixed family $F$ for which the reconfiguration problem is in NP. Proofs can be found in the full version of the paper.

**Theorem 7.** *Reconfiguration with any NPReDAG set of gadgets is in NP.*

**Theorem 8.** *If reconfiguration with some set of gadgets is in NP, than reachability is also in NP.*

### 4.4 Reconfiguration Can Be Easier

In this section we introduce the Labeled Two-Tunnel Single-Use gadget for which the reachability question is harder than the reconfiguration problem. The Labeled Two-Tunnel Single-Use gadget is a DAG gadget where going through either tunnel closes both of them; however, the states are distinguished based on which tunnel was traversed. This is a DAG gadget with a forced distant door closing, so it is NP-complete by Theorem 22 in [4]. We give a polynomial time algorithm for the reconfiguration problem in the full version of the paper.

**Theorem 9.** *Reconfiguration with the Labeled Two-Tunnel Single-Use gadget is in P.*

**Open Problem 3** *Is there a gadget which has different traversability in each state, and reachability with the gadget is NP-complete, but reconfiguration with the gadget problem is in P?*

### 4.5 Reconfiguration with 1-tunnel

Just as with universal traversal, moving to reconfiguration can also be NP-hard with 1-tunnel gadgets. As an example we show that reconfiguration with a diode and a single-use gadget is NP-complete. A diode is a 1-state gadget which only allows traversals from loaction $A$ to location $B$. A single-use gadget is a two state gadget in which state 1 allows a traversal between locations $A$ and $B$ which changes the gadget to state 2. State 2 has no traversals. The reduction is from Hamiltonian path.

**Theorem 10.** *Reconfiguration with the diode and the single-use gadget is NP-complete.*

**Open Problem 4** *For what classes of 1-tunnel gadgets is reconfiguration NP-complete? What about PSPACE-complete?*

**Acknowledgments.** This work grew out of an open problem session and a final project from MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) from Spring 2019.

## References

1. H. A. Akitaya, E. D. Demaine, A. Gonczi, D. H. Hendrickson, A. Hesterberg, M. Korman, O. Korten, J. Lynch, I. Parada, and V. Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *37th International Symposium on Computational Geometry*, 2021.
2. J. Balanza-Martinez, A. Luchsinger, D. Caballero, R. Reyes, A. A. Cantu, R. Schweller, L. A. Garcia, and T. Wylie. Full tilt: Universal constructors for general shapes with uniform external forces. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2689–2708. SIAM, 2019.
3. E. D. Demaine, I. Grosof, J. Lynch, and M. Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 18:1–18:21, La Maddalena, Italy, June 2018.
4. E. D. Demaine, D. Hendrickson, and J. Lynch. 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)*, pages 62:1–62:42, Seattle, Washington, January 2020.
5. R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, 2005.
6. N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.