Traversability, Reconfiguration, and Reachability in the Gadget Framework

Algorithmica
https://doi.org/10.1007/s00453-023-01140-0

Traversability, Reconfiguration, and Reachability in the Gadget Framework

Joshua Ani[1] · Erik D. Demaine[1] · Yevhenii Diomidov[1] · Dylan Hendrickson[1] · Jayson Lynch[1]

Received: 8 April 2022 / Accepted: 23 May 2023

Abstract

Consider an agent traversing a graph of "gadgets", where each gadget has local state that changes with each traversal by the agent according to specified rules. Prior work has studied the computational complexity of deciding whether the agent can reach a specified location, a problem we call ***reachability***. This paper introduces new goals for the agent, aiming to characterize when the computational complexity of these problems is the same or differs from that of reachability. First we characterize the complexity of ***universal traversal***—where the goal is to traverse every gadget at least once—for DAG gadgets (partially), one-state gadgets, and reversible deterministic gadgets. Then we study the complexity of ***reconfiguration***—where the goal is to bring the system of gadgets to a specified state. We prove many cases PSPACE-complete, and show in some cases that reconfiguration is strictly harder than reachability, while in other cases, reachability is strictly harder than reconfiguration.

✉ Joshua Ani
joshuaa@mit.edu

✉ Erik D. Demaine
edemaine@mit.edu

✉ Yevhenii Diomidov
diomidov@mit.edu

✉ Dylan Hendrickson
dylanhen@mit.edu

✉ Jayson Lynch
jaysonl@mit.edu

[1] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Published online: 05 July 2023

1 Introduction

The ***motion-planning-through-gadgets framework***, introduced in [8] and further developed in [2, 3, 5, 9, 10, 13, 15], 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 [10] 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 a target location.[1] In this paper, we extend the gadget model to victory conditions other than reaching a target 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, and are motivated by past uses of the gadgets framework to show hardness of reconfiguration problems and problems with Hamiltonian-path-like constraints. In particular, the gadgets framework has already been used to prove complexity results about reconfiguration problems related to swarm [6] and modular robotics [1], so understanding reconfiguration in the gadgets model may provide an easier and more powerful base for such applications.

More precisely, we first consider the ***universal traversal*** problem of whether the agent can visit every gadget. In Sect. 3, we characterize the complexity of this problem for three classes of k-tunnel gadgets: DAG gadgets (partial characterization), one-state gadgets, and reversible deterministic gadgets. See Table 1. Of particular note is that universal traversal can be harder than reachability with the same gadget.In particular, there are DAG k-tunnel gadgets for which reachability is in NL but universal traversal is NP-complete.

In Sect. 4, we consider the ***reconfiguration*** problem of whether the agent can cause the entire system of gadgets to reach a target configuration. (The configuration does not specify the location of the agent.) Refer to Table 2. We exhibit a gadget with non-interacting tunnels for which reconfiguration is PSPACE-complete, but reachability is in P. For reversible gadgets, we show that reconfiguration is at least as hard as reachability. By contrast, we exhibit a nonreversible gadget for which the reconfiguration is contained in P while reachability is NP-complete.

2 Gadget Model

We now define the gadget model of motion planning, introduced in [8] and expanded upon in [10, 13, 15].

[1] Assembly and motion planning literature often use the term "reachability" to refer to whether an agent can reach a target location. However, reconfiguration literature uses the same term to refer to whether a target state in the configuration space is reachable from another. This would be equivalent to our "reconfiguration" problem, which specifies a target state for every gadget.

Table 1 Summary of our results about universal traversal (visiting every gadget at least once) from Sect. 3, and a comparison to related results about reachability

Gadget Type	Universal Traversal Complexity	Reachability Complexity
DAG Tunnel [§3.1]	NP-complete $\Leftarrow$ true 2-tunnel $\Leftarrow$ distant opening or forced distant closing [Thm. 3.6]; $\in$ P $\Leftarrow$ single-use 1-tunnel [Lem. 3.2]	NP-complete $\Longleftrightarrow$ distant opening or forced distant closing; $\in$ NL otherwise [10]
One-State Tunnel [§3.2]	NP-complete $\Longleftrightarrow$ $\geq$ 3 tunnels & directed; NL-complete $\Longleftrightarrow$ $\leq$ 2 tunnels & directed; $\in$ L $\Longleftrightarrow$ undirected [Thm. 3.12]	$\in$ NL because non-interacting [10]
Reversible Deterministic Tunnel [§3.3]	PSPACE-complete $\Longleftrightarrow$ interacting tunnels; $\in$ NL otherwise [Thm. 3.19]	PSPACE-complete $\Longleftrightarrow$ interacting tunnels; $\in$ NL otherwise [10]

Table 2 Summary of our results about reconfiguration (bringing the system of gadget to a specified state) from Sect. 4, and related results about reachability (some old and some new)

Gadget type	Reconfiguration complexity	Reachability complexity
Reversible [§4.1.1]	PSPACE-complete ⇐ reachability is PSPACE-complete [Thm. 4.1]	PSPACE-complete ⇐ deterministic interacting tunnels [10]
Non-Interacting Box [§4.1.2]	PSPACE-complete [Thm. 4.2]	∈ NL because non-interacting [10]
All Traversals Available [§4.2]	PSPACE-complete sometimes [Cor. 4.4]	∈ L by reduction to undirected graph reachability
Monotonically Opening or Closing [§4.2]	PSPACE-complete sometimes [Cor. 4.4]	PSPACE-complete sometimes [Cor. 4.6 & 4.7]
NPReDAG [§4.3]	∈ NP [Thm. 4.8]	∈ NP [Cor. 4.10]
Labeled Two-Tunnel Single-Use [§4.4]	∈ P [Thm. 4.11]	NP-complete [10]

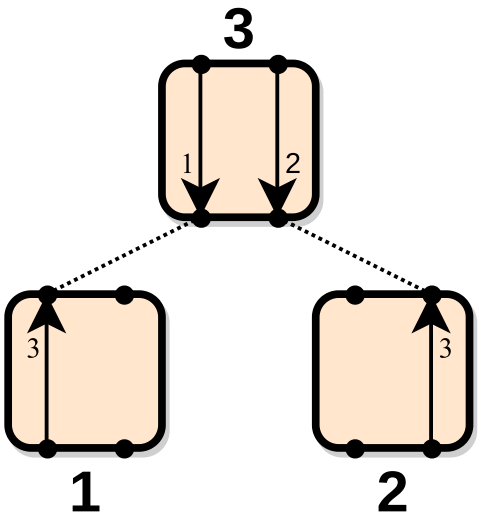


Fig. 1 A diagram describing the locking 2-toggle gadget. Each box represents the gadget in a different state, in this case labeled with the numbers 1, 2, 3. Arrows represent transitions in the gadget and are labeled with the states to which those transition take the gadget. We call state 3 the ***central state*** and states 1 and 2 the ***leaf states***. In the center state, 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

A ***gadget*** G consists of a finite set $Q(G)$ of ***states***, a finite set $L(G)$ of ***locations*** (entrances/exits), and a finite set $T(G)$ of ***transitions*** of the form $(q, a) \to (r, b)$ where $q, r \in Q(G)$ are states and $a, b \in L(G)$ are locations. The transition $(q, a) \to (r, b) \in T(G)$ means that an agent can ***traverse*** the gadget when it is in state q by entering at location a and exiting at location b which changes the state of the gadget from q to r. We use the notation $a \to b$ for a traversal by the agent that does not specify the state of the gadget before or after the traversal. A ***traversal sequence*** $[a_1 \to b_1, \ldots, a_k \to b_k]$ on the locations $L(G)$ is ***legal*** from state q_0 if there is a corresponding sequence of transitions $[(a_1, q_0) \to (b_1, q_1), \ldots, (a_k, q_{k-1}) \to (b_k, q_k)]$, where the start state of each transition matches the end state of the previous transition (q_0 for the first transition).

Equivalently, a gadget is specified by its ***transition graph***, a directed graph whose vertices are state/location pairs, where a directed edge from (q, a) to (r, b) represents that the agent can traverse the gadget from a to b if it is in state q, and that such traversal will change the gadget's state to r. Figure 1 shows an example. Gadgets are ***local*** in the sense that traversing a gadget does not change the state of any other gadgets.

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. (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.

2.1 Decision Problems

Previous work has focused on the reachability problem [8, 10]:

Definition 2.1 For a finite set of gadgets F, ***reachability with 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 introduce and study two new decision problems:

Definition 2.2 For a finite set of gadgets F, ***universal traversal with F*** is the following decision problem. Given a system of gadgets consisting of n copies of gadgets in F and a ***starting 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?

Definition 2.3 For a finite set of gadgets F, ***reconfiguration with F*** is the following decision problem. Given a system of gadgets consisting of n copies of gadgets in F, a ***target state*** for each gadget in the system, and a ***starting location*** in that system of gadgets, is there a path the agent can take from the starting location which puts each gadget in its target state?

A ***configuration*** of a system of gadgets specifies a state for each of the gadgets in the system. We can equivalently think of the reconfiguration problem as consisting of two configurations (initial and target) for the system of gadgets, along with an initial location (but no target location).

2.2 Gadget Types

We will consider several specific classes of gadgets first defined in prior work [8, 10].

Definition 2.4 A ***k-tunnel*** gadget has $2k$ locations, which are partitioned into k pairs called ***tunnels***, such that every transition is between the two locations in the same tunnel.

Definition 2.5 The ***state-transition graph*** of a gadget is the directed graph which has a vertex for each state, and an edge $q \to q'$ for each transition from state q to q'. A ***DAG*** gadget is a gadget whose state-transition graph is acyclic. An ***LDAG*** gadget is a gadget whose state-transition graph would be acyclic if self-loops were removed.

Definition 2.6 A gadget has a ***distant opening*** if there is a transition across a tunnel in some state that ***opens*** a different tunnel, i.e., the tunnel was not traversable in some direction before the transition but becomes traversable in that direction after this transition.

A gadget has a ***forced distant closing*** if there is a traversal across a tunnel in some state and an orientation of some other tunnels such that, for each transition corresponding to the traversal, the traversal ***closes*** some directed traversal in the orientation, i.e., the directed tunnel was traversable before the transition but becomes untraversable after this transition.

DAG gadgets naturally lead to problems with a polynomially bounded number of transitions, because each gadget can be traversed a bounded number of times. Prior work [10] characterizes the complexity of the reachability problem for DAG k-tunnel gadgets: reachability is NP-complete if and only if the gadget has a distant opening or forced distant closing, and in NL otherwise. The same work also characterizes the complexity of the same gadgets in 2-player and team games.

Definition 2.7 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.

Definition 2.8 A gadget is ***reversible*** if every transition can be reversed. More precisely, its transition graph is undirected.

Definition 2.9 A k-tunnel gadget has ***interacting tunnels*** if a transition across one tunnel can change the traversability of another tunnel. That is, there is some transition from state q to state q' across a tunnel $a \to b$ and a different tunnel with locations x and y, such that there is a traversal $x \to y$ in one of the states q and q' but not the other.

Reversible deterministic gadgets are gadgets whose transition graphs are partial matchings, and they naturally lead to unbounded problems. Prior work [10] characterizes the complexity of reachability with reversible deterministic k-tunnel gadgets: reachability is PSPACE-complete if and only if the gadget has interacting tunnels, and in NL otherwise. The same work also partially characterizes the complexity of the same gadgets in 2-player and team games.

3 Universal Traversal

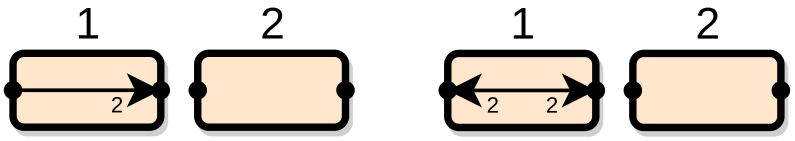
In this section, we consider the question of whether an agent in a system of gadgets can make a traversal across every gadget, called the ***universal traversal*** problem (Definition 2.2).

We provide a full characterization for the complexity of this problem for three classes of gadgets. In Sect. 3.1, we characterize DAG k-tunnel gadgets. In particular, we find that universal traversal is NP-hard for some DAG gadgets for which reachability is in P. Intuitively, this finding is similar to the distinction between finding paths and finding Hamiltonian paths. In Sect. 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, Sect. 3.3 considers the unbounded case by characterizing universal traversal with reversible deterministic k-tunnel gadgets. In this case, the dichotomy is the same as for reachability.

We start with a basic containment result:

Lemma 3.1 *Universal traversal for any gadget is in PSPACE.*

Proof An NPSPACE algorithm repeatedly guesses the next traversal, keeping track of which gadgets have been used, and accepts once they have all been. By Savitch's Theorem [18], universal traversal is in PSPACE. □



(a) Directed single-use tunnel (b) Undirected single-use tunnel

Fig. 2 State diagrams for single-use tunnel gadgets

3.1 DAG Gadgets

In this subsection, we consider universal traversal with k-tunnel DAG gadgets. We find that this problem is NP-hard for any DAG gadget which has and actually uses at least 2 tunnels, in the sense defined below. By contrast, at least some simple 1-tunnel DAG gadgets are in P. Define a ***single-use tunnel*** to be the 1-tunnel gadget with two states: in one state, the tunnel can be traversed in either one direction (***directed***) or both directions (***undirected***); traversal leads to the other state, in which the tunnel cannot be traversed.[2] Figure 2 shows state diagrams for these gadgets.

Lemma 3.2 *Universal traversal with a directed and/or undirected single-use tunnel is in P.*

Proof Universal traversal with a directed and/or undirected single-use tunnel is equivalent to finding an Euler path in the mixed graph resulting from the system of gadgets when we contract each connected component of the connected graph to a single vertex, so every remaining directed or undirected edge corresponds to a gadget's tunnel. When the graph has only undirected edges, this problem can be solved using Euler's characterization of having at most two odd-degree vertices [7, Corollary 4.1]. For directed graphs, the characterization is when all vertices have equal in-degree and out-degree, except for possibly two whose in-degree is 1 larger and smaller respectively than their out-degree [7, Exercise 10.3.2].

For mixed graphs with both directed and undirected edges (corresponding to a mix of directed and undirected single-use tunnels), Papadimitriou [16] gave a polynomial-time algorithm. □

More generally, 1-tunnel DAG gadgets might require traversing the tunnel in a specific pattern of directions, or a set of such patterns. We leave this case open:

Problem 3.3 *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 the sense that it is never possible to make use of multiple tunnels. Figure 3 shows a simple example. We formalize this notion in the following definition.

Definition 3.4 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

[2] Ani et al. [3] call the directed version a "dicrumbler" or "single-use diode".

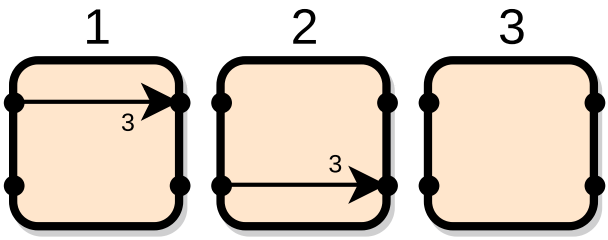


Fig. 3 A 2-tunnel DAG gadget which is not true 2-tunnel

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 3.5 *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.*

What counts as equivalent differs for different victory conditions. For example, any gadget simulation is sufficient to show hardness for reachability, but this may not suffice for universal traversal because traversing the simulation does not necessarily involve traversing every gadget inside it. In the case at hand, the systems of gadgets are equivalent in the sense that the answers to the reachability and universal traversal problems are the same, and the proof should extend to other victory conditions of interest, though not necessarily all of them.

Proof To construct G', we simply collapse the $2k$ locations in G to 2 locations by merging all of the tunnels. Because G is not true 2-tunnel, from any state in G, there is only one tunnel that can ever be traversable. Hence ignoring all of the other tunnels yields the same gadget. If there are different states in G that have different traversable tunnels, we can move them to the same tunnel because these states are never reachable from each other. □

We will use the fact that every ***nontrivial*** DAG gadget—meaning one that has at least one transition in some state—simulates either a directed or an undirected single-use tunnel, by taking a "final" nontrivial state of the gadget [10, Lemma 17].

The rest of this subsection is devoted to proving NP-completeness for universal traversal with true 2-tunnel DAG gadgets.

Theorem 3.6 *Universal traversal with any true 2-tunnel DAG gadget is NP-complete.*

To prove Theorem 3.6, 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

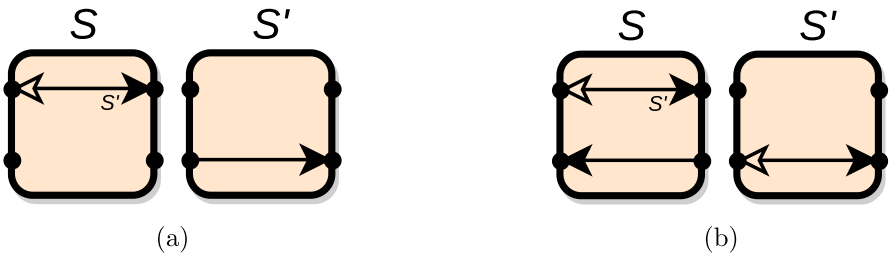


Fig. 4 Two cases for the form of the gadget in Lemma 3.7, assuming traversing the top tunnel to the right opens the bottom tunnel to the right. In (a) the bottom tunnel is not traversable to the left in state q and in (b) it is. Unfilled arrows are traversals that may or may not exist depending on the gadget. Unlabeled transitions may be to arbitrary states not specified here

reached. Such a state exists because the state-transition 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.

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

Lemma 3.7 *Let G be a true 2-tunnel gadget and let q be a final true 2-tunnel state of G. If there exists a transition from q across one tunnel which opens a traversal across another tunnel, then universal traversal with G is NP-hard.*

Proof We will only use the two tunnels involved in the opening transition from q to q' where q' has some traversal which was not possible in q. 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 q has one of the two forms shown in Fig. 4, depending on whether the bottom tunnel can be traversed right to left in q. In either case, the top tunnel may or may not be traversable from right to left in q. Because q is a final true 2-tunnel state, only the bottom tunnel is traversable in S'.

To show NP-hardness of universal traversal with a true 2-tunnel gadget G, we reduce from reachability with G. Because the gadget has a distant opening, reachability is NP-complete [10]. 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 q. We use the construction in either Figs. 5 or 6. 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 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. □

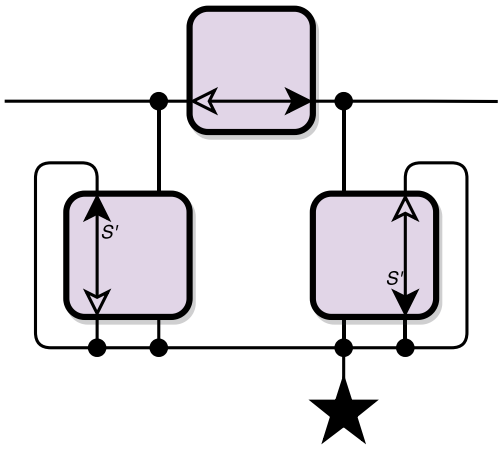


Fig. 5 The construction to allow the agent to use a gadget after reaching the win location, when the bottom tunnel is not traversable in state q (the case of Fig. 4a). The star denotes the goal location

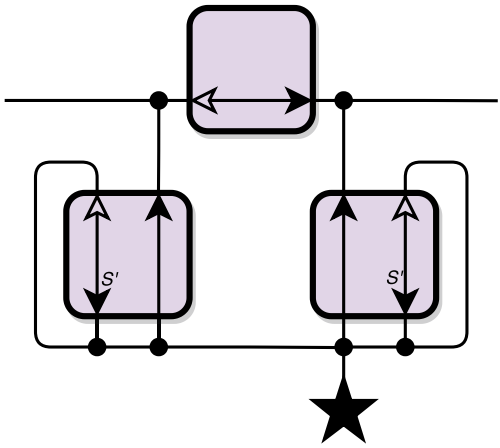


Fig. 6 The construction to allow the agent to use a gadget after reaching the win location, when the bottom tunnel is traversable from right to left in state q (the case of Fig. 4b)

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, only one tunnel 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 3.8 *Let G be a true 2-tunnel gadget and let q be a final true 2-tunnel state of G. Suppose no transition from q across one tunnel opens a traversal across the other tunnel. If, in q, some tunnel can be traversed in one direction but not in the other, then universal traversal with G is NP-hard.*

Proof A directed tunnel with a single-use path on each side is a single-use directed path; because G has a directed tunnel in state q, it simulates a single-use directed path.

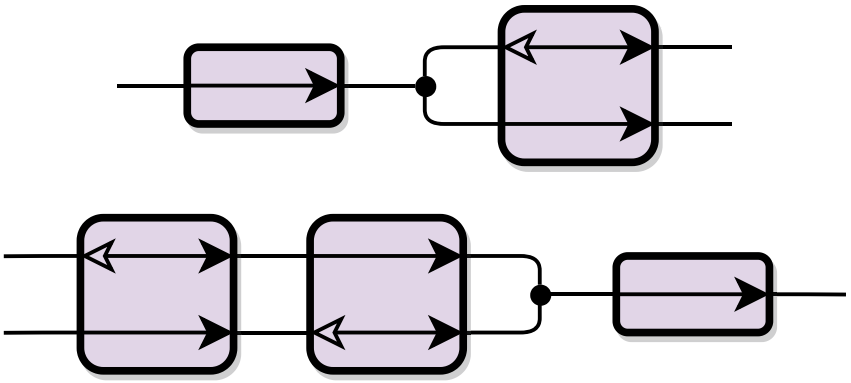


Fig. 7 Vertex gadgets for Lemma 3.8. The first construction is for vertices with in-degree 1 and out-degree 2, and the second construction is for vertices with in-degree 2 and out-degree 1. Each construction contains one or two copies of G in state q and one single-use directed path. By assumption, state q contains two traversable tunnels, at least one of which is directed. If both tunnels are directed, we only need one of the gadgets in state q for the in-degree 2 vertex gadget

We reduce from finding a Hamiltonian path in a directed 3-regular graph with specified start and end vertices s and t [17].[3]

Each vertex of the graph other than s and t becomes one of the vertex gadgets in Fig. 7, depending on its in-degree. We replace s with the right half of the appropriate vertex gadget and t with the left half. The agent begins at s.

If there is a Hamiltonian path, the agent can follow it and thereby make a traversal in every gadget by going through every vertex gadget. Suppose the universal traversal problem is solvable. The agent must use the single-use directed path in each vertex gadget, and thus must go through every vertex. Suppose it enters a vertex gadget with in-degree 2 along the top path, and reaches the vertex in the center. Because, by assumption, making a traversal across a tunnel in q cannot open a traversal on the other tunnel, the bottom tunnel of the left gadget still is not traversable to the left, so the agent cannot exit on the bottom path. Similarly it cannot enter on the bottom path and exit on the top path. Next, the agent cannot enter a vertex gadget with in-degree 1 for the first time on either path on the right, because this would require exiting another vertex gadget to the left on a path it has not used before. If the agent exits a vertex gadget with in-degree 1 on the left, that copy of G is now not true 2-tunnel, so the agent cannot later enter and exit on different paths on the right. Summarizing, the agent always enters vertex gadgets on the left and exits on the right, and it cannot use all three entrances or exits of a vertex gadget. Thus the agent's path corresponds to a path in the graph, and because it must use each single-use directed path this path is Hamiltonian. □

[3] Plesník [17] considers Hamiltonian cycle, but it has many edges that are forced to be included in the cycle (for example, the one on top, or the outgoing edge from any vertex of out-degree 1). If (t, s) is such a forced edge, then there is a Hamiltonian path starting at s and ending at t if and only if there is a Hamiltonian cycle.

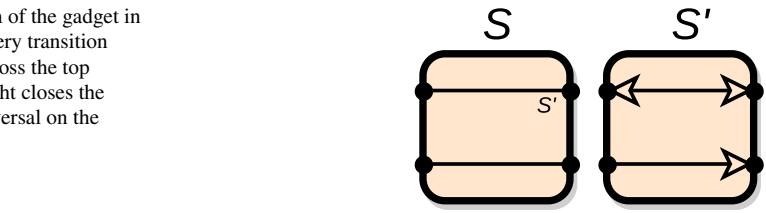


Fig. 8 The form of the gadget in Lemma 3.9. Every transition from state q across the top tunnel to the right closes the right-to-left traversal on the bottom tunnel

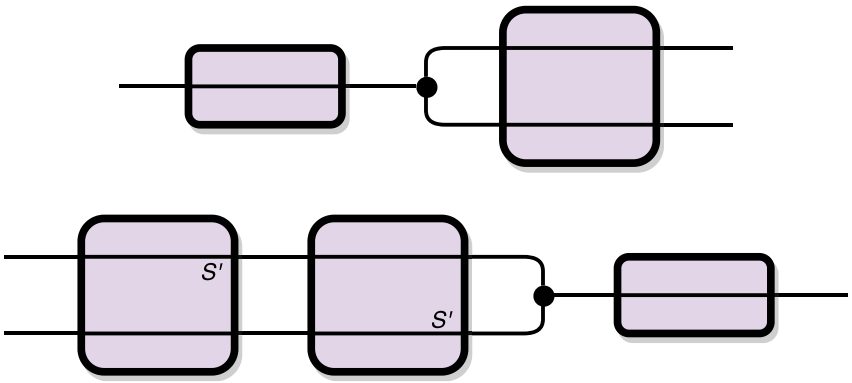


Fig. 9 Vertex gadgets for Lemma 3.9. The first construction is for vertices with in-degree 1 and out-degree 2, and the second construction is for vertices with in-degree 2 and out-degree 1. Each construction contains one or two copies of G in state q and one single-use path

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.

Lemma 3.9 *Let G be a true 2-tunnel gadget and let q be a final true 2-tunnel state of G. Suppose there are two tunnels a and b which can both be traversed in both directions in q. Furthermore, suppose that every transition from q across a from left to right goes to a state in which b cannot be traversed from right to left. Then universal traversal with G is NP-hard.*

Proof The form of state q is shown in Fig. 8. We reduce from finding a Hamiltonian path in a directed 3-regular graph with specified start and end vertices s and t [17], as in Lemma 3.8. We replace each vertex other than s and t with the appropriate vertex gadget in Fig. 9, and replace s and t with the appropriate half of one of these vertex gadgets. If there is a Hamiltonian path, then the agent can follow it to make a traversal in every gadget.

Suppose the agent can make a traversal in every gadget; we consider how it moves through each vertex gadget. It must go across the single-use path in each vertex gadget. Suppose the agent enters a vertex gadget with in-degree 1 on the single-use path. It must exit on a path on the right. If it returns to the vertex gadget along a path on the right, it cannot leave on the other path because at this point that copy of G is not true 2-tunnel; so the agent cannot accomplish anything by returning to the vertex gadget. Now suppose it enters a vertex gadget with in-degree 2 along a path on the left. Because

every transition from q crossing a to the right closes the right-to-left traversal of b, the agent cannot exit the vertex gadget across the other left path. It can return to where it was, or exit across the single-use path.

In particular, by induction the agent must always enter a vertex gadget on one of the in-edges and exit on an out-edge. It cannot use more than two edges on each vertex gadget, and must use the single-use path. So its path corresponds to a Hamiltonian path from s to t. □

Lemma 3.10 *Let G be a true 2-tunnel gadget and let q be a final true 2-tunnel state of G. Suppose there are two tunnels a and b which can both be traversed in both directions in q. Furthermore, suppose that both traversals from state q across a can leave either direction across b traversable, and vice versa. Then universal traversal with G is NP-hard.*

Proof We reduce from finding a Hamiltonian path in an *undirected* 3-regular graph with specified start and end vertices s and t, assuming s and t have degree 1 (so the graph is not quite 3-regular) [11].[4]

We will only use state q and the tunnels a and b. Each vertex of the graph other than s and t is replaced with the construction in Fig. 10, where each of the nine gadgets is in state q and the tunnels involved are a and b. The start location is at s. There is a single-use path leading to t; this forces the agent to end by entering t.

Suppose there is a Hamiltonian path. Then the agent will follow this path through the system of gadgets, and thereby traverse every vertex gadget. At each vertex, by assumption all four traversals across each gadget are currently open. As the agent moves towards the center of the vertex gadget, it will choose transitions so that the path out the edge it intends to exit on stays open; this is possible by assumption. So it is able to follow the Hamiltonian path. Traversing a vertex gadget in this way goes through every gadget in it, so because the path is Hamiltonian the agent uses all of these gadgets. Because the path ends at t, it also uses the single-use path to t.

Conversely, suppose the agent is able to make a traversal in every gadget. It must start at s, and because it traverses the single-use path to t, it must end at t. We will argue that it must pass through each vertex gadget— meaning that it enters along one edge and exits along a different edge— exactly once. Given this claim, we can extract a Hamiltonian path in the graph: consider the sequence of edges between vertex gadgets the agent visits. The agent can only switch edges by passing through a vertex gadget. So once it arrives at an edge via one vertex gadget, it must exit via the other: it cannot backtrack without passing through a vertex gadget multiple times. Thus this sequence of edges gives a walk in the graph from s to t. Because the agent passes through each vertex gadget exactly once, the walk uses each vertex exactly once: in other words, it is a Hamiltonian path.

To show that the agent passes through each vertex gadget exactly once, we will make heavy use of the fact that q is the *final* true 2-tunnel state of G. In particular,

[4] Garey, Johnson, and Tarjan [11] consider Hamiltonian cycle in undirected 3-regular graphs, but it has many edges that are forced to be included in the cycle (every instance of the "required-edge graph" has one such edge). If (x, y) is such a forced edge, then we can delete the edge and add a degree-1 neighbor s to y and a degree-1 neighbor t to x, and there is a Hamiltonian path starting at s and ending at t if and only if there is a Hamiltonian cycle in the original graph.

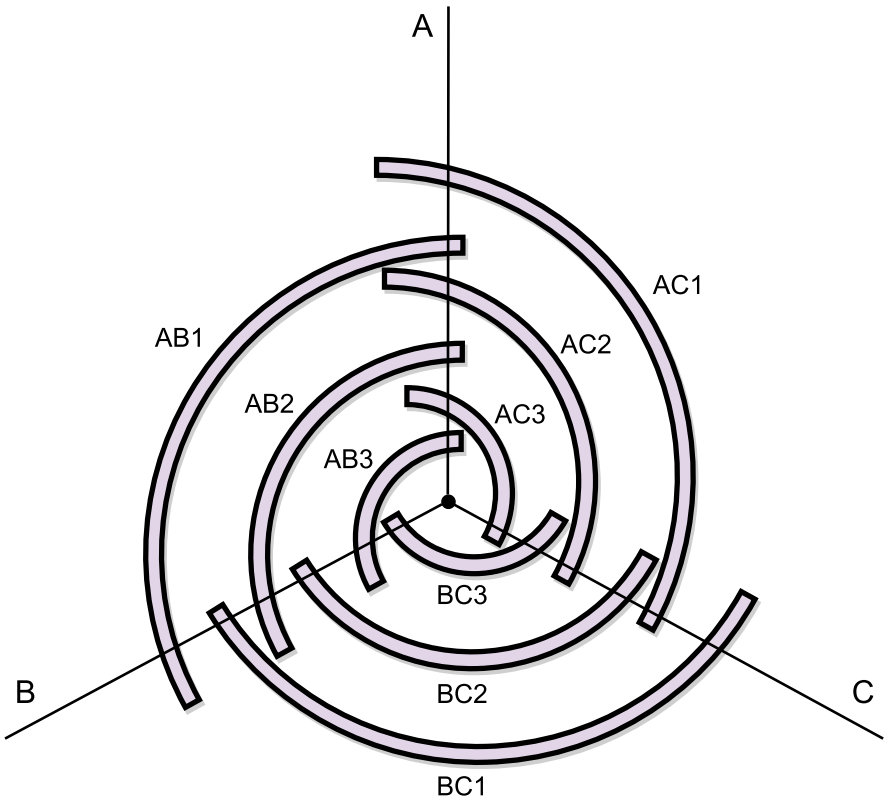


Fig. 10 A vertex gadget for Lemma 3.10. The individual gadgets are true 2-tunnel gadgets and are shown elongated and curved in this diagram. Incident edges are labeled A through C, and gadgets are labeled with the two edges going through them and a number counting from the outside in

after the agent traverses one of the two tunnels on a gadget, it cannot later traverse both tunnels.

Suppose the agent passes through a vertex gadget. It must make a traversal on every individual gadget, leaving them in non-true 2-tunnel states. In order to pass through the vertex gadget a second time, the agent would need to traverse both tunnels on the gadgets which intersect both the entrance and exit edges it uses, but this is impossible because they are not true 2-tunnel anymore.

Now suppose the agent does not pass through some vertex gadget. It may still visit the vertex gadget by entering and exiting along the same edge. We will show that the agent cannot traverse every gadget in the vertex gadget this way. In order to traverse the single-use path to t, the agent must end at t, and in particular cannot end inside the vertex gadget. Consider the first time it traverses one of the innermost gadgets, without loss of generality AC3. Because this is the first such time, the agent must have entered at A and traversed the tunnel of AC3 on A. The agent needs to later exit along A, and thus traverses the tunnels of AC1, AC2, AC3, AB1, and AB2 on A each twice. Because q is final true 2-tunnel, the agent never uses the tunnels of these gadgets on B or C. This

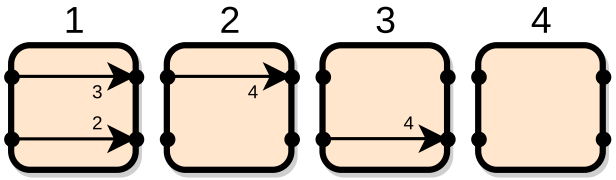


Fig. 11 A DAG gadget for which reachability is in NL but universal traversal is NP-hard. Crossing either directed tunnel closes that tunnel without affecting the other tunnel

mostly cuts off BC2: the only way the agent can make a traversal on BC2 is by entering at A, walking to the tunnel of BC2 on B, turning around, and exiting at A again. In doing so, the agent would use each tunnel of AB3 twice, but this is impossible because q is final true 2-tunnel. Hence it is impossible for the agent to make a traversal in every gadget without passing through the vertex gadget. □

These cases together cover every true 2-tunnel DAG gadget, so we can now prove Theorem 3.6: universal traversal with any true 2-tunnel DAG gadget is NP-complete.

Proof Because the gadget is a DAG, the agent can make a bounded number of traversals in each copy of the gadget. So the solution path has polynomial length, and thus the problem is in NP.

For NP-hardness, we consider a final true 2-tunnel state q and use one of the preceding lemmas. If a transition from q across some tunnel opens a traversal across a different tunnel, NP-hardness follows from Lemma 3.7. Otherwise, if q contains a directed tunnel, we have NP-hardness from Lemma 7. Otherwise, all tunnels traversable in q are traversable in both directions, and no transition from q opens a tunnel. If traversing some tunnel in some direction from q forces the agent to close some traversal across another tunnel, NP-hardness follows from Lemma 3.9. Finally, if there is no traversal with that property, Lemma 3.10 gives NP-hardness. Together these lemmas cover all true 2-tunnel DAG gadgets. □

For a DAG gadget, universal traversal and reachability can have different complexity. Reachability is NP-hard if and only if the gadget has a distant opening or a forced distant closing [10, Theorem 22]. Each of these properties implies that the gadget is true 2-tunnel, so universal traversal is NP-hard whenever reachability is. However, sometimes reachability is in P while universal traversal is NP-hard. For example, for the gadget shown in Fig. 11, NP-hardness of universal traversal is given by Lemma 3.8, whereas reachability is in NL because there are not interacting tunnels.

More generally, the gadgets considered in Lemmas 3.7 and 3.9 all have NP-hard reachability as well as universal traversal, but those considered in Lemmas 3.8 and 3.10 do not necessarily have NP-hard reachability despite universal traversal being NP-hard.

The proofs of Lemmas 3.8, 3.9, and 3.10 can be considered as reductions from finding Hamiltonian paths in planar graphs, which shows the universal traversal problem NP-hard even when restricted to planar systems of gadgets. This leaves open the question of whether this is also true for the gadgets considered in Lemma 3.7.

Problem 3.11 *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 with a k-tunnel gadget that has only one state. Because such gadgets have non-interacting tunnels, reachability with them is in NL [10], but we will see that universal traversal is often NP-complete. This is another example of the distinction between reachability and universal traversal that we saw for DAG gadgets in the previous subsection.

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 because such a tunnel can be removed without affecting the problem. We fully characterize the complexity of universal traversal with such gadgets.

Theorem 3.12 *Let G be a one-state k-tunnel gadget. If G has no directed tunnels, then universal traversal with G is in L. Otherwise, if $k \leq 2$, then universal traversal with G is NL-complete; and if $k \geq 3$, then universal traversal with G is NP-complete.*

We will prove each portion of Theorem 3.12 in a separate lemma.

Lemma 3.13 *Universal traversal with any one-state gadget is in NP.*

Proof If there is a way to use every gadget, this can be done in a number of traversals at most quadratic in the number of tunnels: list the gadgets in an order they can all be visited, and take the shortest path between each pair. The number of gadgets and each such shortest path has length at most the total number of tunnels. So the full solution path can be described in polynomial space. We use this as a certificate; clearly we can check in polynomial time whether a potential solution works. □

Lemma 3.14 *Universal traversal with any one-state k-tunnel gadget with no directed tunnel is in L.*

Proof We solve the universal traversal problem as follows. Iterate over each gadget. For each one, iterate over its locations. For each location, we check whether there is a path from the start location to that location; this is reachability in an undirected graph which can be solved in logarithmic space [19]. If there is a path, we move on to the next gadget. If there is no path, we move on to the next location on the gadget, unless this was the last location, in which case we reject. After finishing all gadgets, we accept.

This algorithm can clearly run in logarithmic space. Because all tunnels are undirected, the agent can visit each gadget in turn and return to the start location after each one. So the agent can use every gadget exactly when there is a path to every gadget from the start location, which is what the algorithm checks. □

Lemma 3.15 *Universal traversal with any one-state k-tunnel gadget with a directed tunnel is NL-hard.*

Proof We reduce from s-t connectivity in directed graphs, which is NL-complete [19]. We will use only one directed tunnel in each gadget.

Given a directed graph with vertices s and t, we first add edges $t \to v$ and $v \to s$ for each vertex v. Then we replace each edge with a directed tunnel in a gadget. This can clearly be done in logarithmic space.

If there is no path from s to t, then the agent can never traverse the tunnel $t \to s$. If there is such a path, the agent can go to t, go to the entrance of a tunnel, go through the tunnel, and return to s. By doing this for each edge in the graph, the agent can make a traversal in every gadget. So the universal traversal problem is solvable exactly when there is a path from s to t. □

Lemma 3.16 *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 spaces with an oracle for reachability in directed graphs. This shows that the universal traversal problem is in $\mathrm{NL}^{\mathrm{NL}}$. We will then explain how the algorithm can be adapted to run in NL. The algorithm first uses the oracle to convert the problem to an instance of 2SAT. It then solves this instance, because 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 L^{NL} algorithm which is known to equal NL [14].

Lemma 3.17 *The 2SAT formula above is satisfiable if and only if the universal traversal problem has a solution.*

Proof First suppose the universal traversal problem is solvable, and consider the assignment which contains the tunnels which are used in the solution. Because 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 $x \to z$. Because each clause $\neg x \vee \neg y$ is satisfied, for any distinct $x, y \in S$ 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.

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.

At this point, we continue with the 2SAT algorithm, because every remaining branch knows the correct value for the input bit we have read. □

Lemma 3.18 *Universal traversal with any one-state k-tunnel gadget with a directed edge is NP-hard when $k \geq 3$.*

Proof Let G be any one-state k-tunnel gadget with a directed edge. We will only use three tunnels in each copy of G, at least one of which is directed. We begin by building the one-state gadget with three directed tunnels. To do this, we connect six copies of G along three paths, such that each path goes through all six copies and the first and last tunnel on each path is directed. Six copies of G is enough to supply these directed edges. The agent can only enter each tunnel of this construction from one side. When it does, it has no choice but to continue all the way through the construction, and in

the process it uses all six gadgets involved. So this simulates the one-state gadget with three directed tunnels, and it suffices to show NP-hardness for this specific gadget.

We prove NP-hardness by a reduction from 3SAT. Each clause becomes a copy of the gadget with three directed tunnels. There are a sequence of branches corresponding to variables, which go through tunnels in the gadgets corresponding to clauses containing the variable or its negation. At the branch corresponding to x, the agent must choose between a path which goes through the gadgets corresponding to clauses with x and a path which goes through gadgets corresponding to clauses with $\neg x$. These paths merge before the branch for the next variable. The start location is at the first branch.

A path through this system of gadgets is exactly an assignment for the formula, and the gadgets visited correspond to satisfied clauses. So it is possible to visit every gadget if and only if the formula is satisfiable. □

Combining Lemmas 3.13 through 3.18, we have Theorem 3.12 characterizing the complexity of universal traversal with one-state k-tunnel gadgets.

3.3 Reversible Deterministic Gadgets

In this subsection, we prove that the complexity of the universal traversal problem for a reversible deterministic k-tunnel gadget is the same as the complexity of the reachability problem for that gadget (as previously characterized in [10]).

Theorem 3.19 *Let G be a reversible deterministic k-tunnel gadget. Then universal traversal (and reachability) with G is PSPACE-complete if G has interacting tunnels, and is in NL otherwise.*

Proof Suppose first that G has no interacting tunnels. Then reachability with G is in NL [10, Theorem 2]. It follows that the question of whether the agent can make a traversal in a target gadget is in NL, because we can solve the reachability question for each usable location of the gadget. To solve universal traversal with G, we check for each gadget whether the agent can use that gadget from the original configuration. If every gadget passes this check, we accept; otherwise we reject.

The output of this algorithm is the AND of the outputs of polynomially many NL algorithms, so it runs in NL. The algorithm works because the agent can follow the path to a gadget, use that gadget, and then reverse its path back to the initial configuration. Doing this for each gadget in series, the agent can make a traversal in every gadget exactly when each gadget can be reached from the initial configuration.

Now suppose G has interacting tunnels. Containment in PSPACE is given by Lemma 3.1. To show PSPACE-hardness, we reduce from reachability with G, which is PSPACE-complete [10, Corollary 7]. By [10, Lemma 5], G simulates a ***1-toggle***, the gadget whose state diagram is shown in Fig. 12. Furthermore, when the agent traverses the simulation of the 1-toggle, it visits every gadget in the simulation; this is clear from examining the proof of this lemma.

Given an instance of the reachability problem for G, we modify it by adding a 1-toggle from the win location to each location in the system of gadgets. We also add a gadget at the win location which is usable if and only if the agent reaches the win location. This can clearly be done in polynomial time.

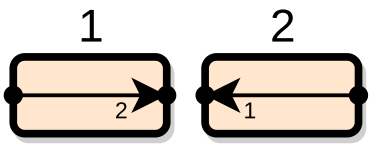


Fig. 12 The state diagram of the 1-toggle

If the agent can reach the win location, then it can travel along one of these 1-toggles, cross back and forth across a gadget, and return along the 1-toggle. In this way, the agent can make a traversal in every gadget, including those in the simulated 1-toggles and the gadget added at the win location. Conversely, in order to traverse every gadget, the agent most traverse the gadget at the win location, which requires being able to reach the win location in the original reachability instance. Thus the reachability instance is solvable if and only if the constructed universal traversal instance is solvable. □

4 Gadget Reconfiguration

In this section, we study the question of whether an agent can traverse a system of gadgets to bring the system to a target configuration, called the ***reconfiguration*** problem (Definition 2.3. In Sect. 4.1, we show that, with reversible deterministic gadgets, reconfiguration is at least as hard as reachability. On the other hand, we give an example of a reversible nondeterministic gadget with non-interacting tunnels for which the reconfiguration problem is PSPACE-complete, whereas reachability with any gadget with non-interacting tunnels is always in NL [10, Theorem 2]. Section 4.2 shows some methods for constructing new PSPACE-complete gadgets from known ones, and uses these methods to show that the reconfiguration problem can be PSPACE-complete even when a gadget does not change traversability, while reachability can be PSPACE-complete even when a gadget monotonically opens or monotonically closes tunnels. Finally, in Sect. 4.3, we show an interesting connection between reconfiguration and bounded reachability problems, expanding the classes of gadgets we know of for which these problems are in NP. We also exhibit a gadget with which reachability is NP-complete but reconfiguration is in P.

4.1 Reconfiguring Reversible Gadgets

In this subsection, we first show that, for reversible gadgets, reconfiguration is at least as hard as reachability. We then exhibit a reversible deterministic gadget with non-interacting tunnels for which the reconfiguration problem is PSPACE-complete, providing an example where reconfiguration is harder than reachability.

4.1.1 Reconfiguration is as Hard as Reachability

Theorem 4.1 *Let S any set of reversible gadgets where at least one has a transition that changes state. There is a polynomial-time reduction from reachability with S to reconfiguration with S.*

Proof We use the same trick as the one used to show reconfiguration Nondeterministic Constraint Logic is PSPACE-complete [12]. Given an instance of reachability, at the win location we add a gadget with a state-changing transition and a loop which allows the agent to take this transition. 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 the one the transition leads to. If the reconfiguration problem has a solution, the agent must be able to visit the added gadget, so the reachability problem has a solution. Conversely, if the reachabality problem has a solution, the agent can perform it, make the transition in the added gadget, and then take the inverse of all transitions of the reachability solution (in reverse order). This is possible and leaves all gadgets in their target state because the gadgets are reversible. □

The assumption that some gadget is nontrivial is a very minor one, but it isn't strictly needed: if no gadget in S has a state-changing transition, then reconfiguration with S is trivial: the only obtainable configuration is the starting one. Reachability with S reduces to reachability in a directed graph, and thus is in NL.[5] Since both problems are in P, there are polynomial-time reductions in both directions except for the degenerate case where the answer is always the same such is if every gadget in S has only one state so reconfiguration is always possible.

4.1.2 PSPACE-Complete Reversible Deterministic Gadget with Non-Interacting Tunnels

There are cases where the reconfiguration problem is strictly harder. In this section we describe a reversible deterministic gadget with non-interacting tunnels, so the reachability problem is in NL [10, Theorem 2], but for which the reconfiguration problem is PSPACE-complete.

The ***Non-Interacting Box gadget*** is a reversible, deterministic, 12-state, two-tunnel gadget shown in Fig. 13. Each tunnel can be traversed either once or twice consecutively in the same direction, depending on the state. Although going through one tunnel never changes the traversability of the other tunnel, it does sometimes change how many times the other tunnel can be traversed in the same direction. We will refer to the four rightmost states (B, C, G and H) as the ***right square***, and the four bottommost states (I, J, K, and L) as the ***bottom square***.

A key fact about the Non-Interacting Box gadget is that it is ***balanced***, meaning the net number of traversals across each tunnel is determined by the change in state. In other words, we can arrange the states in a grid, where B and E, F and I, and G and J are each in the same grid cell, such that every transition moves one grid cell in the direction of the traversal. The coordinates of the grid cell track the net number of traversals across each tunnel.

Theorem 4.2 *Reconfiguration with the Non-Interacting Box gadget is PSPACE-complete.*

[5] For reversible gadgets with no state-changing transitions, traversals must be undirected, so in fact this is in L.

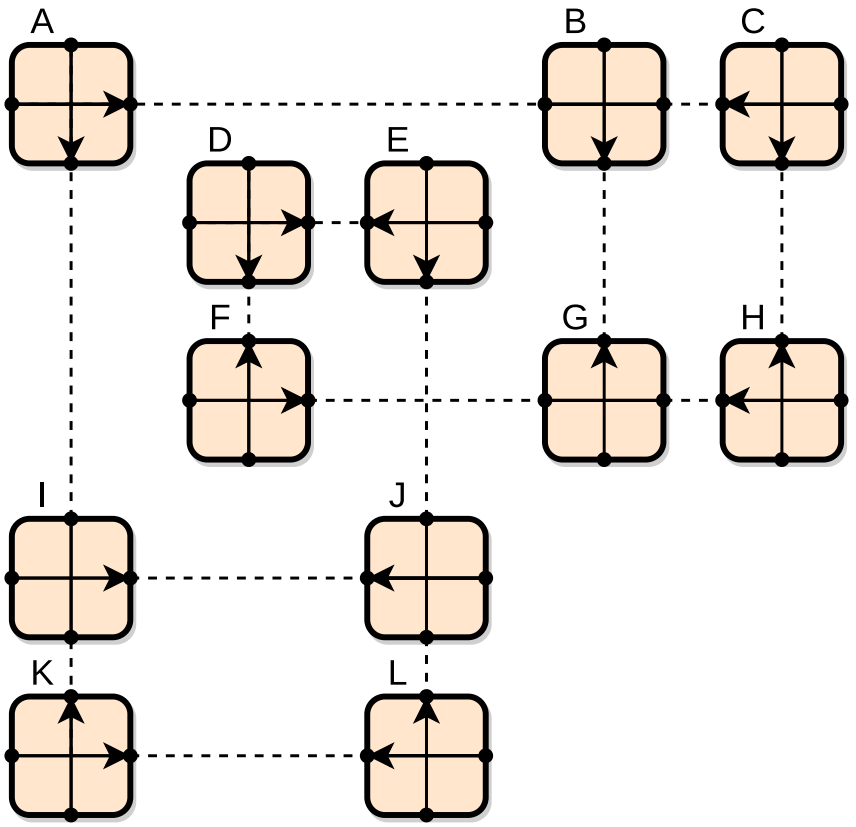


Fig. 13 The state diagram of the Non-Interacting Box gadget. Dashed edges indicate reversible pairs of transitions when crossing the gadget in the same direction: for instance, a left-to-right traversal in state A leaves the gadget in state B, and a right-to-left traversal in state B leaves the gadget in state A. All Non-Interacting Box gadgets will be drawn in the same orientation

Proof To show PSPACE-completeness, we will reduce through two new decision problems. First, ***targeted reconfiguration*** is a slight modification of reconfiguration first defined in [13], where we also require the agent to end in a designated target location as in reachability. Theorem 4.1 can easily be adapted to prove targeted reconfiguration with the locking 2-toggle hard, simply by making the target location the same as the start location.

In ***multi-agent motion planning***, there are multiple agents in the system of gadgets. The agents perform a sequence of transitions, each of which involves only one agent. There is no limit on the number of agents at a location, but our constructions will ensure that there are at most two in the same place simultaneously. ***Cooperative targeted reconfiguration*** is the following decision problem about multi-agent motion planning: given a system of gadgets, a specified starting count of agents at each location, a target configuration, and a target count of agents at each location, is the a sequence of transitions resulting in the target configuration with the correct target counts?

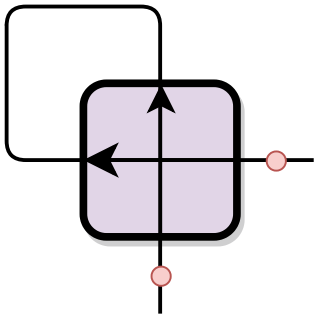


Fig. 14 The multi-agent 1-toggle. The two helper agents are denoted by red dots. The Non-Interacting Box gadget is in either state L or H, depending on the state of the simulated 1-toggle

The second step in our chain is a reduction to cooperative reconfiguration with the Non-Interacting Box gadget, where each location is forced (by construction) to have at most two agents simultaneously. Finally, we will reduce this problem to (single-agent) reconfiguration with the Non-Interacting Box gadget, by enabling one agent to simulate the actions of many.

Multi-agent 1-Toggle. We will begin by simulating a multi-agent version of the 1-toggle, a gadget whose state diagram is shown in Fig. 12. A regular 1-toggle can be easily constructed from the Non-Interacting Box gadget by taking a single tunnel in an appropriate state. Instead, we will build a gadget that does not allow individual agents through at all, but if it has an agent on either side of it, a third agent can use the gadget as though it were a 1-toggle.

To build our multi-agent 1-toggle we simply connect the tunnels together as shown in Fig. 14 It will have ***helper agents*** standing ready at the two entrances. When the Non-Interacting Box gadget is in state L or H (labeled in Fig. 13, the simulated 1-toggle can be traversed bottom-to-right or right-to-bottom, respectively.

Because the Non-Interacting Box gadget is balanced, we can determine the change in positions of agents just from the change in state. So it suffices to consider the reachable states for different numbers of agents present. We consider starting in state L; state H is similar. With only the two helper agents, the only reachable states are the bottom square, depending on which subset of helper agents have moved 'inside' the gadget. Since the reachable set of states are a simple grid, the state is entirely determined by the net number of traversals across each tunnel. In particular, the helper agents are not able to leave the multi-agent 1-toggle in the same direction, since doing so would require a net of one upwards and one rightwards (or one leftwards and one downwards) traversal (the helper agents can switch places, but this doesn't accomplish anything).

If a third agent arrives on the right when the Non-Interacting Box gadget is in state L (or equivalently, any configuration the helper agents can reach from L), this agent does not allow anything new. This is because having an addition agent able to traverse right-to-left does not make any additional states reachable from the bottom square.

If instead the third agent arrives on the bottom, then the 1-toggle can be used: the new agent can move inside, putting the Non-Interacting Box gadget in state J. Now the two helper agents can freely switch the gadget between states J and G, by both entering and then both exiting, going through either D or A. This takes advantage of

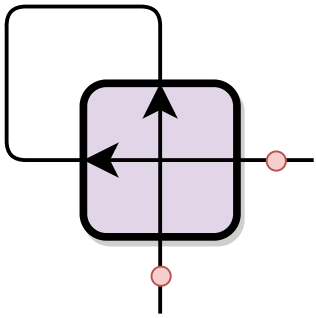


Fig. 15 The multi-agent locking 2-toggle in the center state. Red dots denote helper agents. The middle gadget is a Non-Interacting Box gadget in state A, and the other four gadgets are multi-agent 1-toggles

the fact that these traversals do not commute, which is represented by the fact that the relevant portion of the state diagram is not a grid. Once the gadget is in state G, the third agent can exit to the right, leaving it in H. Every path from L to H has a net one upwards and one rightwards traversal, so by switching from from state L to H the net effect is one agent entering the bottom and exiting the right. Since no state is further right than H, once the gadget is in H it is not possible for an agent to move from the bottom to the right.

To summarize: when there is at most one agent at each entrance, nothing interesting happens, but if there are at least two at one entrance and one at the other, the gadget can be used as a 1-toggle. The additional agents are needed to go around a noncommuting square in the state diagram.

Multi-agent Locking 2-Toggle. The multi-agent locking 2-toggle will be comprised of one Non-Interacting Box gadget, four multi-agent 1-toggles (which do not contain helper agents, and are drawn with a wide arrow), and six helper agents. It will allow an additional agent to interact with it as though it were a locking 2-toggle. Two helper agents will be located in the horizontal and vertical connections next to the Non-Interacting Box gadget, and the other four agents will be external, each adjacent to one of the multi-agent 1-toggles. Note, these external four agents will be shared between gadgets rather than duplicated.

The center state (state 3 in Fig. 1) is shown in Fig. 15 where the Non-Interacting Box gadget is in state A. Without any additional agents, the helper agents cannot do much: the multi-agent 1-toggles prevent any additional agents from getting inside. The two agents with access to the Non-Interacting Box gadget can change its state in a cycle of 8 states in a doubly-covered square, but are unable to go anywhere else. Another agent arriving at the right or bottom does not allow any new behavior because of the multi-agent 1-toggles.

If an agent arrives at the top, it is able (with help) to cross the first 1-toggle, both agents can move down through the Non-Interacting Box gadget (leaving it in state K), and then the two agents (and the agent at the bottom) can work together to have one of them move to the bottom. An agent has traversed the simulated locking 2-toggle, leaving it in a leaf state. The resulting configuration is shown in Fig. 16. The other leaf state is similar.

From this configuration, if an agent arrives on the left, it could move through the multi-agent 1-toggle, but only one agent would be able to pass the Non-Interacting

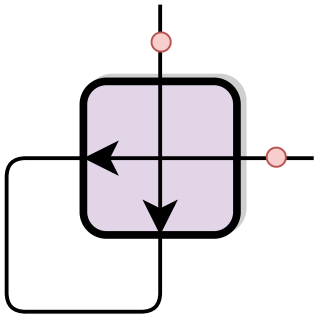


Fig. 16 The multi-agent locking 2-toggle in the leaf state after traversing top-to-bottom. Red dots denote helper agents. The middle gadget is a Non-Interacting Box gadget in state K, and the other four gadgets are multi-agent 1-toggles

Box gadget: having a net two traversals to the right requires going through state A. Thus it would not be able to pass the second 1-toggle. So a left-to-right traversal is now impossible, as is needed for our simulation of a locking 2-toggle. All other traversals other than reversing the one taken to get to the leaf state are prevented by the multi-agent 1-toggles.

Hardness for Cooperative Targeted Reconfiguration. We reduce from reconfiguration with locking 2-toggles, by replacing each locking 2-toggle with our construction of a multi-agent locking 2-toggle in the appropriate state. We place the two helper agents inside each multi-agent locking 2-toggle, and one helper agent in each connected component of the original connection graph. Note that adjacent multi-agent locking 2-toggles share external helper agents. We place an additional agent at the start location. The target configuration and agent counts are defined similarly.

If we ignore the agent at the start location, there is exactly one agent that as access to each side of each multi-agent 1-toggle; possibly involving crossing the Non-Interacting Box gadget inside a multi-agent locking 2-toggle. In particular, no agent can cross a multi-agent 1-toggle.

The agent at the start location changes this, since there are now two agents there, which is enough to activate a multi-agent 1-toggle. Because of the behavior of our constructions, by induction it will always be the case that each side of each multi-agent 1-toggle has one agent that can reach it, except for one that has two. The position of the two agents represents the agent in the single-agent reconfiguration problem. This doubled agent can navigate the system in exactly the same way as an agent navigating the original system of locking 2-toggles. So a solutions to the original targeted reconfiguration problem correspond to solutions to the cooperative targeted reconfiguration problem, though we may need to return helper agents to the appropriate locations (without changing the states of simulated gadgets) at the end.

Simulating Extra Agents. Now we wish to simulate the multi-agent reduction with a single agent. We can directly build a (single agent) 1-toggle out of the Non-Interacting Box gadget, by using the vertical tunnel in states C and H. Recall that our reduction ensured that no connected component of the connection graph has more than two agents at any given point in time. Starting with such an instance of cooperative targeted reconfiguration with the Non-Interacting Box gadget, we attach two 1-toggles to each connected component of the connection graph, each representing a potential agent.

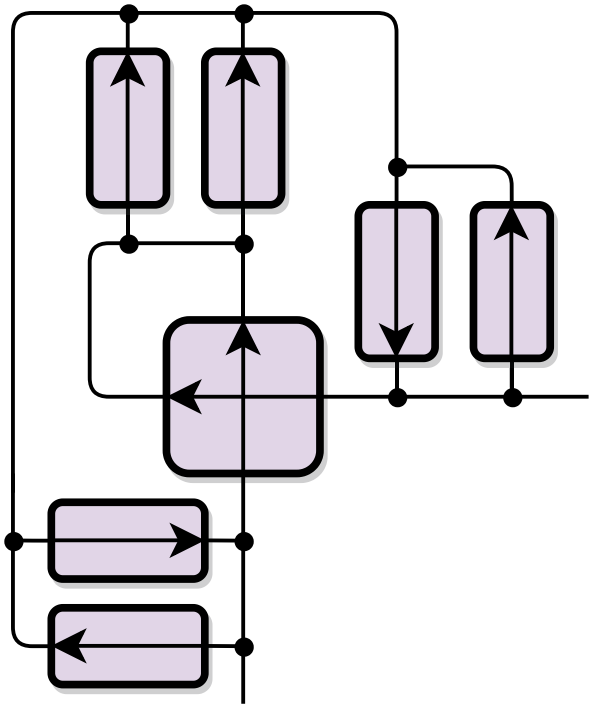


Fig. 17 Our construction for a single agent to simulate multiple using 1-toggles, applied to the multi-agent 1-toggle (Fig. 14). The middle gadget is an Non-Interacting Box gadget, and other gadgets are 1-toggles. The path along the top and left edges is the hub. 1-toggles pointing away from the hub represent agents

The other end of all the one toggles are all connected to a ***hub***, which is where the agent starts. For each connected component, the number of added 1-toggles directed towards it is the number of agents that start there. An example of this transformation is shown in Fig. 17.

From the hub, the agent is able to cross a 1-toggle to 'embody' the agent it represents in the multi-agent problem. The agent is then in the same location and able to interact with original instance exactly how the embodied agent would. If the agent then traverses a 1-toggle back to the hub, the virtual agent is now at the component that 1-toggle connects to. This 'remembers' where the virtual agent is, and allows the real agent to embody it again later. Because the multi-agent problem never has more than two agents in a connected component, the two 1-toggles we added are sufficient to record all virtual agents. When the agent is at the hub, the configuration of the entire network corresponds to the configuration of the multi-agent network plus the number of agents in each component, except that there are two ways to represent one agent in a component. To resolve this ambiguity and complete the reduction, we can pick a consistent one of the 1-toggles to flip when representing the presence of a single agent. □

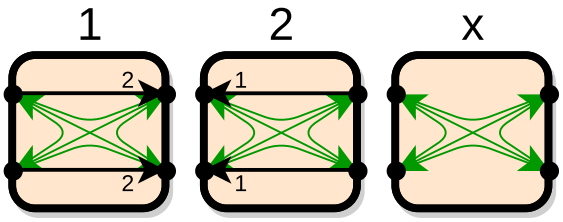


Fig. 18 The simply breakable 2-toggle. Green arrows are transitions to x

4.2 PSPACE-Complete Monotonically Opening and Closing Gadgets

In this subsection, we will demonstrate hard gadgets in some classes which may seem as though they should only contain easy gadgets. The main idea is to construct a gadget which behaves well when used like a known-hard gadget, but might also have other transitions which are allowed but put the gadget into some undesirable state. [6]

Let G be any gadget. Define a ***breakable*** G to be a gadget obtained from G by adding any number of new states, which we call ***broken***, and then adding any collection of transitions to broken states. Many different gadgets are breakable Gs, including G itself. As suggested by the terminology, one should think of a breakable G as "breaking" when the agent takes a transition to a broken state; we will arrange that the game cannot be won if this happens, making breaking the gadget effectively illegal.

The most useful breakable G is the ***simply breakable*** G, which has one broken state x, and has every possible breaking transition: it adds the transition $(a, q) \to (b, x)$ for all locations a and b and states q (including x). For example, Fig. 18 shows the simply breakable 2-toggle, where the ***2-toggle*** is the reversible deterministic 2-tunnel gadget obtained by removing state x and the green arrows from Fig. 18. Reachability with the 2-toggle is PSPACE-complete [8], so by Theorem 4.1 reconfiguration is as well. While a simply breakable gadget allows any sequence of traversals, if the agent does anything that would not be allowed by the base gadget, the gadget becomes permanently stuck in the broken state.

Theorem 4.3 *For any gadgets G and G' where G' is a breakable G, there is a polynomial-time reduction from reconfiguration with G to reconfiguration with G'.*

Proof We simply replace each copy of G in a system with a copy of G' in the same state, and use the same target states. If the original instance has a solution, it is still a solution when each G is replaced with a breakable G. If, in the system of G's, the agent makes any transition to a broken state, that gadget can never reach its target state, so the instance becomes unsolvable. Thus any solution to this reconfiguration problem never puts any copy of G' in a broken state, and so is also a solution to the original instance. □

For the remainder of this subsection, let $\widetilde{G}$ be any gadget for which both reachability and reconfiguration are PSPACE-complete. Any such gadget (for example, the 2-toggle) suffices for our results.

[6] The conference version of this paper [4] introduced "shadow states" and "verifiable gadgets", which were a major inspiration for checkable gadgets in [3]. We have reframed these results to better align with the terminology of checkable gadgets.

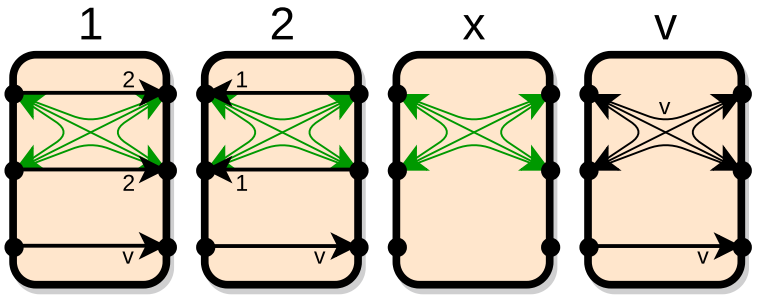


Fig. 19 The simplify verifiable 2-toggle. Green arrows are transitions to x

Corollary 4.4 *There is a gadget that never changes its traversability, and indeed all traversals are available in every state, and for which reconfiguration is PSPACE-complete.*

Proof Because reconfiguration with $\widetilde{G}$ is PSPACE-complete, so is reconfiguration with the simply breakable $\widetilde{G}$. The simply breakable $\widetilde{G}$ has all traversals available in every state, and thus is such a gadget. □

By contrast, reachability with a gadget that never changes its traversability is in NL because it reduces to directed graph reachability. When all tunnel traversals are possible, reachability is in L because the graph is undirected.

Next we use these ideas to prove reachability PSPACE-complete for some new classes of gadget.

For a gadget G, a ***verifiable*** G is built from a breakable G by adding an unbroken state v, two locations c_{in} and c_{out}, transitions $(c_{in}, q) \to (c_{out}, v)$ for each unbroken state q, and all transitions $(a, v) \to (b, v)$ where a and b are locations of G. Call the traversal $c_{in} \to c_{out}$ the ***checking traversal***; intuitively, making this traversal verifies that the gadget was not previously broken, because it is possible only in an unbroken state (including v). The verifiable G built from the simply breakable G is called the ***simplify verifiable*** G. Figure 19 shows the simplify verifiable 2-toggle.

Compared to the checkable gadgets framework [3], a verifiable G is a special case of a "simply checkable G" (which is itself a special case of "postselection"), and our notions of broken states coincide. Our next theorem is a direct consequence of these relations and the checkable gadgets framework, but we provide a self-contained proof which is simpler because we do not need as much generality.

Theorem 4.5 *For any gadgets G and G' where G' is a verifiable G, there is a polynomial-time reduction from reachability with G to reachability with G'.*

Proof Consider a system of Gs with start and target locations s and t. Replace each copy of G with a copy of G' in the same state. Add a path that goes from t through the checking traversal on each gadget in series, and finally to a location t'. Consider reachability on this system with start and target locations s and t'.

Any solution for the original reachability problem gives a solution for this one: after reaching t, we can proceed through all of the checking traversals because no copy of

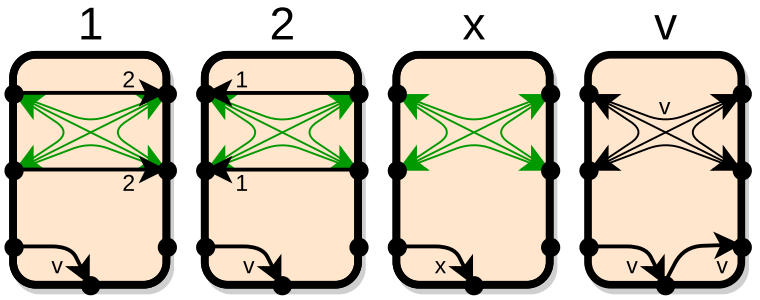


Fig. 20 The gadget constructed for Corollary 4.7, using a 2-toggle for $\widetilde{G}$. The bottom three locations are c_{in}, c, and c_{out} from left to right. Green arrows are transitions to x

G' is in a broken state. Conversely, a solution for the new system must reach t' by passing through $c_{in} \to c_{out}$ on every copy of G' after reaching t. In order to do so, no gadget can be in a broken state, so the portion of the solution up to arriving at t is a solution to the original problem. □

Monotonically Opening and Closing Gadgets. A gadget is ***monotonically opening*** if its traversability never decreases: if there is a transition from state q to r, and the traversal $a \to b$ is available in q, then $a \to b$ is also available in r. Similarly, a gadget is ***monotonically closing*** if its traversability never increases: if there is a transition from state q to r, and the traversal $a \to b$ is *not* available in q, then $a \to b$ is also not available in r.

We now use verifiable gadgets to show that there are both monotonically opening gadgets and monotonically closing gadgets with which reachability is PSPACE-complete. This may be surprising because the number of changes of traversability in such a system of gadgets is bounded, so one might suspect reachability with such gadgets to fall in NP.

Corollary 4.6 *There exists a monotonically closing gadget for which reachability is PSPACE-complete.*

Proof Because reachability with the $\widetilde{G}$ is PSPACE-complete, so is reachability with the simplify verifiable $\widetilde{G}$. The only way the simplify verifiable $\widetilde{G}$ changes traversability is when the checking traversal closes because it enters a broken state. In particular, the simplify verifiable $\widetilde{G}$ is monotonically closing. □

Corollary 4.7 *There exists a monotonically opening gadget for which reachability is PSPACE-complete.*

Proof We construct a gadget G similar to the simplify verifiable $\widetilde{G}$, but with two sequential checking traversals. Specifically, starting with the simply breakable $\widetilde{G}$, add a state v and three locations c_{in}, c, and c_{out}. Then add transition $(c_{in}, x) \to (c, x)$ and the transitions $(c_{in}, q) \to (c, v)$ for all states q other than x, and add transition $(c, v) \to (c_{out}, v)$ and all transitions $(a, v) \to (b, v)$ where a and b are locations of G. Figure 20 shows the resulting gadget when $\widetilde{G}$ is the 2-toggle. Note that $c_{in} \to c$ is traversable in every state.

Gadget G is monotonically opening: the only change in traversability is that $c \to c_{out}$ becomes available after traversing $c_{in} \to c$ from an unbroken state.

Gadget G also simulates the simplify verifiable $\widetilde{G}$, simply by ignoring c. Then the only possible use of the new locations is following $c_{in} \to c \to c_{out}$, which is legal from every state except x, exactly matching the simplify verifiable $\widetilde{G}$. Thus, using the proof of Corollary 4.6, reachability with G is also PSPACE-complete. □

4.3 Reconfiguration and DAG-like Gadgets

Past work studies DAG gadgets [10] and LDAG gadgets [15] as naturally bounded classes of gadgets, which leads to reachability questions in NP. We now define a related generalization and describe cases in which the reachability question remains in NP.

Given a gadget G, a ***DAG-like decomposition*** is a partition of the states $Q(G)$ into k clusters $Q_1, Q_2, \ldots, Q_k$ such that, if we take the state-transition graph of G and combine the vertices within in each cluster Q_i, we obtain a directed acyclic graph on the k cluster vertices. For each cluster Q_i, define the ***induced gadget*** G_i to be the gadget consisting of states in Q_i and all transitions $(q, a) \to (r, b)$ of G for which $q, r \in Q_i$. Intuitively, G consists of k subgadgets $G_1, G_2, \ldots, G_k$ connected by a DAG-like structure. Specifically, we call G ***F-DAG-like*** if every induced gadget G_i comes from the family of gadgets F. For example, DAG gadgets are F-DAG-like where F is the family of trivial gadgets with no transitions, and LDAG gadgets are F-DAG-like where F is the family of single-state gadgets (which can have only loops in the state-transition graph).

Given that both DAG gadgets and LDAG gadgets have reachability in NP, we may wonder more generally for what gadget families F the F-DAG-like gadgets have reachability or reconfiguration in NP. We initially believed that this might be true for any gadgets F with non-interacting tunnels, but this is not the case. Consider the gadget in Fig. 20. We can perform a DAG-like decomposition with two clusters of states, $\{1, 2, x\}$ and $\{v\}$. Each induced gadget does not have interacting tunnels, and in fact does not change its traversability. But we showed in Corollary 4.7 that reachability with this gadget is PSPACE-complete.

On the positive side, we show that, if F is a family of gadgets for which the reconfiguration problem is in NP, then reconfiguration and reachability with F and with F-DAG-like gadgets are all also in NP. Define a ***NPReDAG gadget*** to be an F-DAG-like gadget where F is a family of gadgets for which the reconfiguration problem is in NP.

Theorem 4.8 *Reconfiguration with NPReDAG gadgets is in NP.*

Proof We give the following certificate for 1-player motion planning with an NPReDAG gadget. We list all of the DAG-like transitions taken in the solution and the states of all of the gadgets before and after the transition. Further, for each pair of adjacent DAG-like transitions we imagine the reconfiguration problem on the system of gadgets which is only comprised by the reconfigurable super-node gadgets and takes this system from the state after the last DAG-like transition to the state before the next DAG-like transition. This problem is solvable in NP by definition, so we

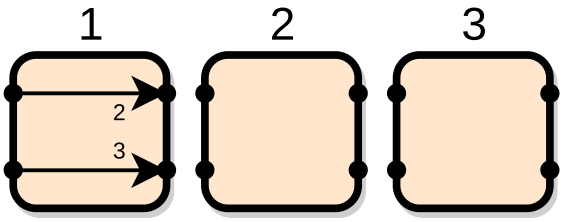


Fig. 21 The labeled two-tunnel single-use gadget

provide each of these certificates. The verifier can now check in polynomial time that the final state is the target state, that the polynomial many DAG-like transitions are valid transitions and take the given pre-transition state to the post-transition state, and that the (polynomially many) portions of the path between the DAG-like transitions have some valid path performing that reconfiguration. □

Theorem 4.9 *Reachability is in NP for gadgets where reconfiguration is in NP.*

Proof We now essentially want to "guess" the final configuration that the system will be in when the agent solves the reconfiguration problem and then solve the reconfiguration problem. However, this strategy also needs to verify that the agent actually reaches the win location. To do this first we take the reachability instance and add at the win location a loop with a gadget that has access to a transition with a state change. If there is none, then both reconfiguration and reachability are trivially in NL. To be able to change the state of the added gadget, an agent must have reached the location of the loop. Thus we will take as a certificate a final configuration of the system of gadgets which has the added gadget in a different state, as well as the certificate for the reconfiguration problem from the initial state to this new target state. □

Corollary 4.10 *Reachability with NPReDAG gadgets is in NP.*

4.4 Reconfiguration can be Easier

In this subsection, we introduce the Labeled Two-Tunnel Single-Use gadget, shown in Fig. 21 with which reachability is harder than reconfiguration. This gadget is a DAG gadget where going through either tunnel of the gadget 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 [10, Theorem 22]. We now give a polynomial-time algorithm for the reconfiguration problem.

Theorem 4.11 *Reconfiguration motion planning with the Labeled Two-Tunnel Single-Use gadget is in P.*

Proof Call states 2 and 3 ***terminal states***. Now consider what the initial and final configurations of the gadgets can look like. If the initial state is terminal, then the gadget cannot be traversed. Similarly, if the initial and final configuration are both state 1, then the gadget cannot have been traversed, because there is no way to return the gadget to state 1 after traversal. Thus the only case we need to consider is starting in state 1 and ending in a terminal state. In this case, the labeling of the target state

(2 or 3) tells us which of the two tunnels must have been traversed to reach that state. We can thus construct the directed graph which contains only those tunnels and ask whether there is a path traversing them all exactly once. Because this problem is exactly checking for the existence of an Eulerian path in a directed graph, we can solve it in polynomial time [7, Exercise 10.3.2], [16]. □

It would be interesting to have an example of a gadget with different traversability in every state, so that the easiness of reconfiguration with it would not be using a degeneracy which is indistinguishable to reachability.

Acknowledgements This work was initiated 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. A preliminary version of the paper appeared at WALCOM 2022 [4].

Funding 'Open Access funding provided by the MIT Libraries'

Declarations

Conflict of interest We have no conflicts of interest to declare.

References

1. Akitaya, H.A., Demaine, E.D., Gonczi, A., Hendrickson, D.H., Hesterberg, A., Korman, M., Korten, O., Lynch, J., Parada, I., Sacristán, V.: Characterizing universal reconfigurability of modular pivoting robots. In: 37th International Symposium on Computational Geometry (2021)
2. Ani, J., Bosboom, J., Demaine, E.D., Diomidov, Y., Hendrickson, D., Lynch J.: Walking through doors is hard, even without staircases: proving PSPACE-hardness via planar assemblies of door gadgets. In: Proceedings of the 10th International Conference on Fun with Algorithms (FUN 2020), pp 3:1–3:23 (2020)
3. Ani, J., Chung, L., Demaine, E.D., Diomidov, Y., Hendrickson, D., Lynch, J.: 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)
4. Ani, J., Demaine, E.D., Diomidov, Y., Hendrickson, D.H., Lynch, J.: Traversability, reconfiguration, and reachability in the gadget framework. In: Mutzel, Petra, Rahman, M.d. Saidur, Slamin, (eds.), Proceedings of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM 2022), volume 13174 of Lecture Notes in Computer Science, pp 47–58, Jember, Indonesia (2022)
5. Ani, J., Demaine, E.D., Hendrickson, D., Lynch, J.: Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. In: Mutzel, Petra, Rahman, Md. Saidur, Slamin (eds.), Proceedings of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM 2022), volume 13174 of Lecture Notes in Computer Science, pp 187–198, Jember, Indonesia, March 24–26 (2022)

6. Balanza-Martinez, J., Luchsinger, A., Caballero, D., Reyes, R., Cantu, A.A., Schweller, R., Garcia, L.A., Wylie, T.: Full tilt: Universal constructors for general shapes with uniform external forces. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, pp 2689–2708. SIAM (2019)
7. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. North-Holland (1976)
8. 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), pp 18:1–18:21, La Maddalena, Italy, (2018)
9. Demaine, E.D., Hearn, R.A., Hendrickson, D., Lynch, J.: PSPACE-completeness of reversible deterministic systems. In: Proceedings of the 9th Conference on Machines, Computations and Universality (MCU 2022), pp 91–108, Debrecen, Hungary, August–September (2022)
10. 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 (2020)
11. Garey, M.R., Johnson, D.S., Tarjan, R.E.: The planar Hamiltonian circuit problem is NP-complete. SIAM J. Comput. **5**(4), 704–714 (1976)
12. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theor. Comput. Sci. **343**(1–2), 72–96 (2005)
13. Hendrickson, D.: Gadgets and gizmos: A formal model of simulation in the gadget framework for motion planning. Master's thesis, Massachusetts Institute of Technology (2021)
14. Immerman, Neil: Nondeterministic space is closed under complementation. SIAM J. Comput. **17**(5), 935–938 (1988)
15. Lynch, J.: A framework for proving the computational intractability of motion planning problems. PhD thesis, Massachusetts Institute of Technology (2020)
16. Papadimitriou, C.H.: On the complexity of edge traversing. J. ACM **23**(3), 544–554 (1976)
17. Plesník, J.: The NP-completeness of the Hamiltonian cycle problem in planar diagraphs with degree bound two. Inf. Process. Lett. **8**(4), 199–201 (1979)
18. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. **4**(2), 177–192 (1970)
19. Wigderson, A.: The complexity of graph connectivity. In: Havel, Ivan M., Koubek, Václav (eds.), Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS 1992), pp. 112–132, Prague, Czechoslovakia (1992)