Reconfiguring Undirected Paths

Erik D. Demaine[⋆], David Eppstein[⋆⋆], Adam Hesterberg[⋆⋆⋆], Kshitij Jain[†], Anna Lubiw[‡], Ryuhei Uehara[§], and Yushi Uno[¶]

Abstract. We consider problems in which a simple path of fixed length, in an undirected graph, is to be shifted from a start position to a goal position by moves that add an edge to either end of the path and remove an edge from the other end. We show that this problem may be solved in linear time in trees, and is fixed-parameter tractable when parameterized either by the cyclomatic number of the input graph or by the length of the path. However, it is PSPACE-complete for paths of unbounded length in graphs of bounded bandwidth.

1 Introduction

In this paper, we consider the problem of sliding a fixed-length simple path within an undirected graph from a given starting position to a given goal position. The path may move in steps where we add an edge to either end of the path and simultaneously remove the edge from the opposite end, maintaining its length. Effectively, this can be thought of as sliding the path one step along its length in either direction. The allowed movements of the path are similar to those of trains in a switchyard, or of the model trains in any of several train shunting puzzles; the edges of the path can be thought of as the cars of a train. However, unlike train tracks, we do not constrain connections at junctions of track segments to be smooth: a path that enters a vertex along an incident edge can exit the vertex along any other incident edge. Additionally, we do not distinguish the two ends of the path from each other.

Our aim is to understand the computational complexity of two natural reconfiguration problems for such paths: the *decision problem*, of testing whether it is possible to reach the goal position from the start position, and the *optimization*

[⋆] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, edemaine@mit.edu

[⋆⋆] Computer Science Department, University of California, Irvine, Irvine, CA 92697, USA, eppstein@uci.edu. Supported in part by NSF grants CCF-1618301 and CCF-1616248.

[⋆⋆⋆] MIT Mathematics Department, 77 Massachusetts Ave., Cambridge, MA 02139, USA, achesterberg@gmail.com

[†] University of Waterloo, Canada, k22jain@uwaterloo.ca

[‡] University of Waterloo, Canada, alubiw@uwaterloo.ca

[§] Japan Advanced Institute of Science and Technology, Nomi, Japan, uehara@jaist.ac.jp

[¶] Graduate School of Engineering, Osaka Prefecture University, Japan, uno@cs.osakafu-u.ac.jp

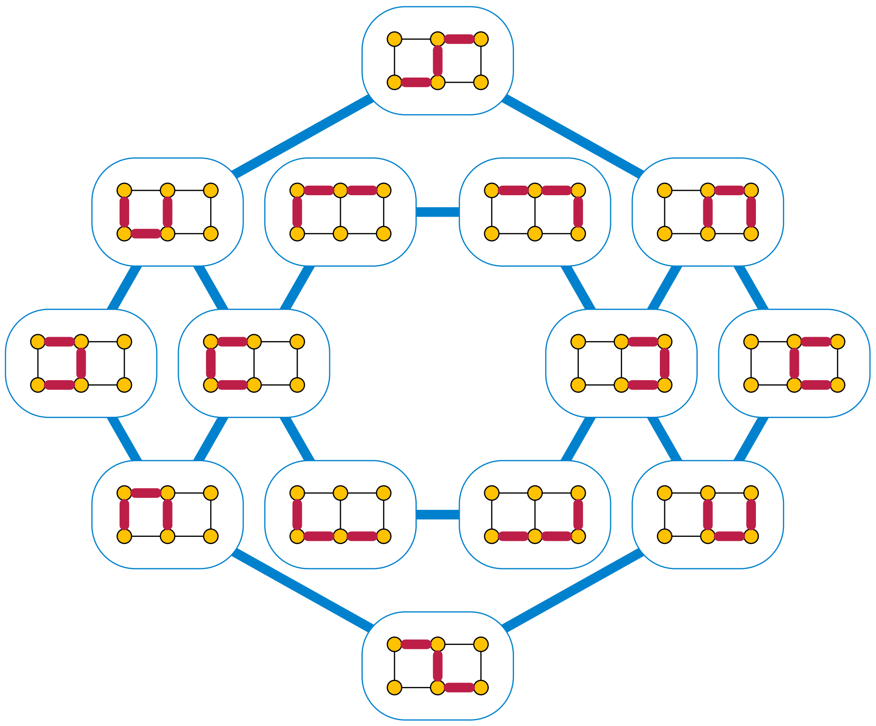

Fig. 1. State space of three-edge paths on a six-vertex graph

problem, of reaching the goal from the start in as few moves as possible. One natural upper bound for the complexity of these problems is the size of the state space for the problem, a graph whose vertices are paths of equal length on the given graph and whose edges represent moves from one path to another (Figure 1). If a given graph has N paths of the given length, and M moves from one path to another, we can solve either the decision problem or the optimization problem in time $O(M + N)$ (after constructing the state space) by a simple breadth-first search. As we will see, it is often possible to achieve significantly faster running times than this naive bound. On the other hand, the general problem is hard, even on some highly restricted classes of graphs.

Specifically, we prove the following results:

1. The decision problem for path reconfiguration is fixed-parameter tractable when parameterized by the length of the path. This stands in contrast to the size of the state space for the problem which (for paths of length k in n-vertex graphs) can have as many as $\Omega(n^{k+1})$ states.
2. For paths of unbounded length in graphs parameterized by the circuit rank, both the decision and the optimization problems can be solved in fixed-parameter tractable time by state space search. The same problem can

be solved in polynomial (but not fixed-parameter tractable) time when parameterized by feedback vertex set number.

3. The optimization problem for path reconfiguration in trees can be solved in linear time, even though the state space for the problem has quadratic size.
4. The decision problem for path reconfiguration is PSPACE-complete for paths of unbounded length, even when restricted to graphs of bounded bandwidth. Therefore (unless $\mathsf{P} = \mathsf{PSPACE}$) path reconfiguration is not fixed-parameter tractable when parameterized by bandwidth, treewidth, or related graph parameters.

Because of limited space, the detailed versions of several of our results are deferred to the full version of this paper (arXiv:1905.00518).

1.1 Related work

There has been much past research on reconfiguring structures in graphs, with motivations that include motion planning, understanding the mixing of Markov chains and bounding the computational complexity of popular games and puzzles. See, for instance, Ito et al. [1] for many early references, and Mouawad et al. [2] for more recent work on the parameterized complexity of these problems. Often, in these problems, one considers moves in which the structure changes by the removal of one element and the addition of an unrelated replacement element (token moving) or in which an element of the structure changes only locally, by moving along an edge of the graph (token sliding).

Several authors have considered problems of reconfiguring paths or shortest paths under token jumping or token sliding models of reconfiguration [3–5]. However, the path sliding moves that we consider are different. Token sliding moves only a single vertex or edge of a path along a graph edge, while we move the whole path. And although our path sliding moves can be seen as a special case of token jumping, because they remove one edge and add a different edge, token jumping in general would allow the replacement of edges or vertices in the middle of a path, while we allow changes only at the ends of the path.

The path reconfiguration problem that we study here is also closely related to a popular video game, Snake, which has a very similar motion to the path sliding moves that we consider. Our problem differs somewhat from Snake in that we consider bidirectional movement, while in Snake the motion must always be forwards. Snake is typically played on grid graphs, and it is known to be PSPACE-complete to determine whether the Snake can reach a specific goal state from a given start state on generalized grid graphs [6]. Independently of our work, Gupta et al [7] have found that reconfiguring snakes (paths that can move only unidirectionally) is fixed-parameter tractable in the length of the path, analogously to our Theorem 1.

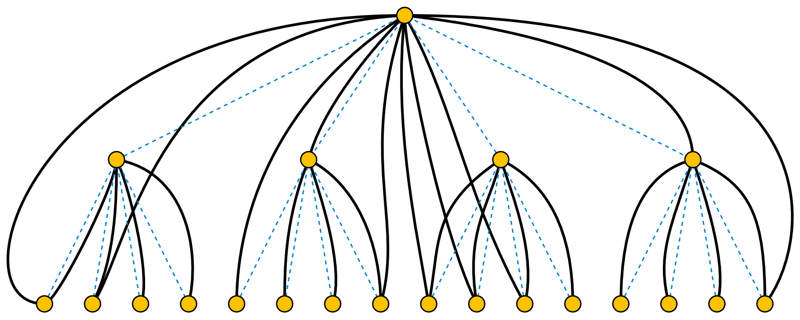

Fig. 2. A graph G of tree-depth 2 (solid black edges) and a tree T realizing this depth (dashed blue edges).

2 Preliminaries

2.1 Reconfiguration sequences and time reversal

Definition 1. *We define a* reconfiguration step *in a graph G to be a pair of edges (e, f), and a* reconfiguration sequence *to be a sequence σ of reconfiguration steps. We may* apply *a reconfiguration step to a path P by adding edge e to P and removing edge f, whenever f is one of the two edges at the ends of P, e is incident to the vertex at the other end, and the result of the application is another simple path. We may apply a reconfiguration sequence to a path by performing a sequence of applications of its reconfiguration steps. If applying reconfiguration sequence σ to path P produces another path Q we say that we can reconfigure P into Q or that σ takes P to Q.*

If (e, f) is a reconfiguration step, then we define its time reversal *to be the step (f, e). We define the time-reversal of a reconfiguration sequence σ to be the sequence of time reversals of the steps of σ, taken in the reverse order. If σ takes P to Q, then its time reversal takes Q to P. For this reason, when we seek the existence of a reconfiguration sequence (the path reconfiguration decision problem) or the shortest reconfiguration sequence (the path reconfiguration optimization problem), reconfiguring a path P to Q is equivalent under time reversal to reconfiguring Q to P. We call this equivalence* time-reversal symmetry*.*

We define the length *$|P|$ of a path P to be its number of edges, and the length $|\sigma|$ of a reconfiguration sequence to be its number of steps.*

2.2 Tree-depth

Tree-depth is a graph parameter that can be defined in several equivalent ways [8], but the most relevant definition for us is that the tree-depth of a connected graph G is the minimum depth of a rooted tree T on the vertices of G such that each edge of G connects an ancestor-descendant pair of T (Figure 2). Here, the depth of a tree is the length of the longest root-to-leaf path. Another way of expressing the connection between G and T is that T is a depth-first search tree for a supergraph of G. For disconnected graphs one can use a forest in place of a tree, but we will only consider tree-depth for connected graphs.

Tree-depth is a natural graph parameter to use for path configuration, because it is closely connected to the lengths of paths in graphs. If a graph G has maximum path-length ℓ, then clearly its tree-depth can be at most ℓ, because any depth-first search tree of G itself will achieve that depth. In the other direction, a graph with tree-depth d has maximum path-length at most $2^{d+1} - 2$, as can be proven inductively by splitting any given path at the vertex closest to the root of a tree T realizing the tree-depth. Therefore, the tree-depth and maximum path-length are equivalent for the purposes of determining fixed-parameter tractability. The parameterized complexity of reconfiguration problems on graphs of bounded tree-depth has been studied by Wrochna [9]. However, these graphs are highly constrained, so algorithms that are parameterized by tree-depth are not widely applicable.

We will prove as a lemma that path reconfiguration is fixed-parameter tractable for the graphs of bounded tree-depth. Because these graphs have bounded path lengths, this result will be subsumed in our theorem that path reconfiguration is fixed-parameter tractable when parameterized by path-length. However, we will use this lemma as a stepping-stone to the theorem, by proving that in arbitrary graphs we can either find a structure that allows us to solve the problem easily or restrict the input to a subgraph of bounded tree-depth.

3 Parameterized by path length

In this section we show that path reconfiguration is fixed-parameter tractable when parameterized by path length. As discussed above, our strategy is to find a structure (*loose paths*, defined below), whose existence allows us to solve the reconfiguration problem directly. When these structures do not exist or exist but cannot be used, we will instead restrict our attention to a subgraph of bounded tree-depth. We begin with the lemma that the problem is fixed-parameter tractable when parameterized by tree-depth instead of path length.

3.1 Tree-depth

Our method for graphs of low tree-depth is based on the fact that, when these graphs are large, they contain a large amount of redundant structure: subgraphs that are all connected to the rest of the graph in the same way as each other. When this happens, we can eliminate some copies of the redundant structures and reduce the problem to a smaller instance size.

Definition 2. *Given a graph G and a vertex set S, we define an S-*flap *to be a subset X of the vertices of G such that X is disjoint from S and there are no edges from X to $G \setminus \{S \cup X\}$. We say that two S-flaps X and Y are* equivalent *when the induced subgraphs $G[S \cup X]$ and $G[S \cup Y]$ are isomorphic, by an isomorphism that reduces to the identity mapping on S (Figure 3).*

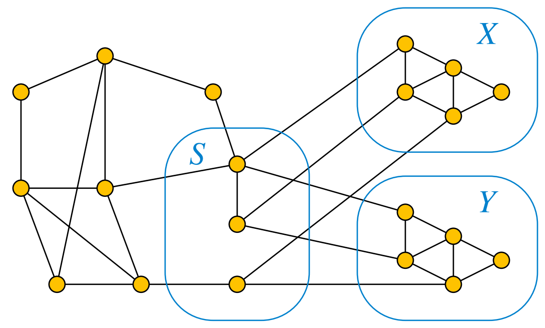

Fig. 3. Two equivalent S-flaps X and Y in a graph G

Observation 1 *For any graph G and any vertex set S, a path of length k can include vertices from at most $\lceil (k-1)/2 \rceil$ S-flaps of G.*

Proof. The path has $k+1$ vertices, and any two vertices in distinct flaps must be separated by at least one vertex of S. $\square$

Lemma 1. *Suppose we are given an instance of path reconfiguration for paths of length k in a graph G, and that G contains a subset S that is disjoint from the start and goal positions of the path and has more than $\lceil (k+1)/2 \rceil$ pairwise equivalent S-flaps $X_1, X_2, \ldots,$ all disjoint from the start and goal. Then we can construct an equivalent and smaller instance by removing all but $\lceil (k+1)/2 \rceil$ of these equivalent S-flaps.*

Proof. Any reconfiguration sequence in the original graph can be transformed into a reconfiguration sequence for the reduced graph by using one of the remaining S-flaps whenever the sequence for the original graph enters an S-flap. Because the S-flaps are equivalent, the moves within the flap can be mapped to each other by the isomorphism defining their equivalence, and by Observation 1 there will always be a free S-flap to use in the reduced graph. $\square$

Lemma 2. *We can solve the decision or optimization problems for path reconfiguration in time that is fixed-parameter tractable in the tree-depth of the input graph.*

Proof. We provide a polynomial-time kernelization algorithm that uses Lemma 1 to reduce the instance to an equivalent instance whose size is a function only of the given tree-depth d. The problem can then be solved by a brute-force search on the resulting smaller instance. We assume without loss of generality that we already have a tree decomposition T of depth d, as it is fixed-parameter tractable to find such a decomposition when one is not already given [8, p. 138]. Recall that, for graphs of tree-depth d, the length k of the paths being reconfigured can be at most $2^{d+1} - 2$.

We apply Lemma 1 in a sequence of stages so that, after stage i, all vertices at height i in T have $O(1)$ children. As a base case, for stage 0, all vertices at height 0 in T automatically have 0 children, because they are the leaves of T. Therefore, suppose by induction on i that all vertices at height less than i in T have $O(1)$ children.

For a given vertex v at height i, let S_v be the set of ancestors of v in T (including v itself). Then, for each child w of v in T, let X_w be the set of descendants of w (including w itself). Then X_w is an S_v-flap, because S_v includes all of its ancestors in T and it can have no edges to vertices that are not ancestors in T. If we label each vertex in T by the set of heights of its adjacent ancestors, then the isomorphism type of $G[S_v \cup X_w]$ is determined by these labels, so two children u and w of T have equivalent S_v-flaps whenever they correspond to isomorphic labeled subtrees of W. Trees of constant size with a constant number of label values can have a constant number of isomorphism types, so there are a constant number of equivalence classes of S_v flaps among the sets W_x. Within each equivalence class, we apply Lemma 1 to reduce the number of flaps within that equivalence class to a constant. After doing so, we have caused the vertices of T at height i to have a constant number of children, completing the induction proof.

To implement this method in polynomial time, we can use any polynomial time algorithm for isomorphism of labeled trees [10]. The equivalence of subtrees of T by labeled isomorphism may be finer than the equivalence of the corresponding subgraphs of G by graph isomorphism (because two different labeled trees may correspond to isomorphic subgraphs) but using the finer equivalence relation nevertheless leaves us with a kernel of size depending only on d. The time for this algorithm can be bounded by a polynomial, independent of the parameter. □

As the following observation shows, this result is nontrivial in the sense that its time bound is significantly smaller than the worst-case bound on the size of the state space for the problem.

Observation 2 *In graphs of tree-depth d, the number of paths of a given length can be $\Theta(n^{2^d})$.*

Proof. Let T be a tree realizing the depth of the given graph. To prove that the number of paths is $O(n^{2^d})$, consider the vertex v in any path that is highest in tree T, and apply the same bound inductively for the two parts of the path on either side of v, both of which must live in lower-depth subtrees. The total number of paths can be at most the product of the numbers of choices for these two smaller paths.

To prove that the number of paths can be $\Omega(n^{2^d})$, let T be a star as the base case for depth one (with $\Omega(n^2)$ paths of length two) and at each higher depth connect two inductively-constructed subtrees through a new root vertex v. Given a tree T constructed in this way, let G be the graph of all ancestor-descendant pairs in T (Figure 4). Each two paths in the two subtrees can be connected to

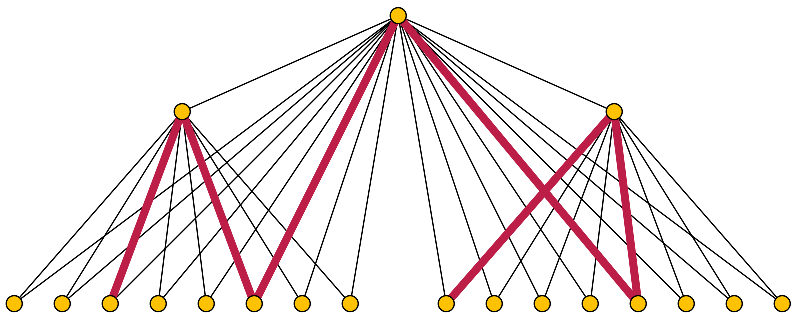

Fig. 4. One of $\Omega(n^4)$ paths of length 6 in a graph of tree-depth 2

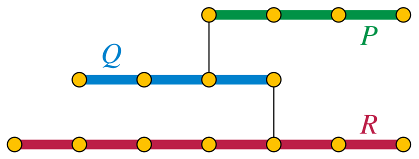

Fig. 5. A loose path R for start and goal paths P and Q

each other through v, so the number of paths in the whole graph is the product of the numbers of paths in the two subtrees. □

Therefore, an algorithm that searched the entire state space would only be in XP, not FPT.

3.2 Loose paths

We have seen that graphs without long paths are easy for path reconfiguration. Next, we show that graphs with long paths are also easy. The following definition is central to this part of our results:

Definition 3. *Consider an instance of path reconfiguration consisting of a graph G, a start path P of length k, and a goal path Q of length k. We define a* loose path *to be a simple path R of length $2k$ in G, such that R is vertex-disjoint from both P and Q (Figure 5).*

Lemma 3. *Let R be a loose path for an instance (G, P, Q) of path reconfiguration, such that it is possible to reconfigure path P into a path that uses at least one vertex of R. Then for every vertex v in R, it is possible to reconfigure path P into a sub-path of R for which v is an endpoint.*

Proof. Consider a sequence σ of reconfiguration steps starting from P that results in a path using at least one vertex of R and is as short as possible. Because σ is as short as possible and R is disjoint from P, the last move of σ must cause exactly one vertex u of R to be an endpoint of the reconfigured path. Because

R has length $2k$, at least one endpoint of R is at distance k or more along R from u. By sliding the path along R towards this endpoint, we can reconfigure it so that it lies entirely along R. Again, because R has length $2k$, one of the two sub-paths of R ending at v has length at least k. By concatenating to σ an additional sequence of steps that slide the path along R (if necessary) we can reconfigure the starting path so that it lies within this sub-path and ends at v. $\square$

We call a loose path R that meets the conditions of Lemma 3 a *reachable loose path*.

Lemma 4. *If an instance (G, P, Q) of path reconfiguration has a reachable loose path, and the graph G is connected, then all loose paths for that instance are reachable.*

Proof. Let R be a reachable loose path, and L be any other loose path. If R and L share a vertex v, then it is possible to slide any sub-path of R so that it includes this vertex, showing that L meets the conditions of Lemma 3. If R and L are disjoint, let T be a shortest path between them in G, and let v be the unique vertex of T that belongs to R. By Lemma 3, we can reconfigure the starting path so that it lies along R and ends at v. From there, we can slide the path along T until it reaches the other endpoint of T, a vertex of L. This shows that L meets the conditions of Lemma 3. $\square$

It will be helpful to bound the tree-depth of graphs with no loose path.

Observation 3 *If an instance of path reconfiguration for paths of length k has no loose path, then its graph has tree-depth less than $4k$.*

Proof. Form a depth-first-search forest F of the subgraph formed by removing all vertices of the start and goal paths. Because there is no loose path, F has depth at most $2k - 1$. Form a single rooted path R of the vertices of the start and goal paths, in an arbitrary order. Connect R and F into a single tree T (not necessarily a subtree of the input graph) by making each root of F be a child of the leaf node of R. Then every edge in the given graph connects an ancestor–descendant pair in T, because either it connects two vertices in the depth-first-search forest or it has at least one endpoint on the ancestral path R. Thus, T meets the condition for trees realizing the tree-depth of a graph, and its depth is at most $4k - 1$, so the given graph has tree-depth at most $4k - 1$. $\square$

3.3 Win-win

We show now that we can either restrict our attention to a subgraph of bounded tree-depth or find a reachable loose path, in either case giving a structure that allows us to solve path reconfiguration.

Definition 4. *Given an instance (G, P, Q) of path reconfiguration, we say that S is a* reachable *set of vertices if, for every vertex v in S, there exists a sequence* *of reconfiguration steps that takes P into a path that uses vertex v. We say that S is an* inescapable *set of vertices if, for every vertex v that is not in S, there does not exist a sequence of reconfiguration steps that takes P into a path that uses vertex v.*

Lemma 5. *Given an instance (G, P, Q) of path reconfiguration, parameterized by the length k of the start and goal paths, we can in fixed-parameter-tractable time find either a reachable loose path, or a reachable and inescapable set S of vertices that induces a subgraph $G[S]$ of tree-depth at most $4k - 1$.*

Proof. We will maintain a vertex set S that is reachable and induces a subgraph of tree-depth less than $4k$ until either finding reachable loose path or finding that S is inescapable and has no path. Initially, S will consist of all vertices of the start path P; clearly, this satisfies the invariants that S is reachable and has tree-depth less than $4k$.

Then, while we have not terminated the algorithm, we perform the following steps:

– For each edge uv where $u \in S$ and $v \notin S$, use the algorithm of Lemma 2 to test whether P can be reconfigured within $S \cup \{v\}$ (a graph of tree-depth at most $4k$) into a path that uses vertex v. If we find any single edge uv for which this test succeeds, we go on to the next step. Otherwise, if no edge uv passes this test, S is inescapable and we terminate the algorithm.
– Test whether the graph $S \cup \{v\}$ contains a loose path. Finding a path of fixed length is fixed-parameter tractable for arbitrary graphs [11–13] and can be solved even more easily by standard dynamic programming techniques for graphs of bounded tree-depth. If this test succeeds, the loose path must contain v, as the remaining vertices have no loose path. In this case, we have found a reachable loose path (as v is reachable) and we terminate the algorithm.
– Add v to S and continue with the next iteration of the algorithm. Because (in this case) v is reachable but $S \cup \{v\}$ contains no loose path, it follows that including v in S maintains the invariants that S be reachable and induce a subgraph with tree-depth at most $4k - 1$.

Because each iteration adds a vertex to S, the loop must eventually terminate, either with a reachable inescapable subgraph of low tree-depth (from the first step) or with a reachable loose path (from the second step). □

3.4 Fixed-parameter tractability

We are now ready to prove our main result:

Theorem 1. *The path reconfiguration decision problem is fixed-parameter tractable when parameterized by the length of the start and goal paths.*

Proof. Our algorithm for path reconfiguration begins by applying Lemma 5 to find either a reachable inescapable subgraph of low tree-depth or a reachable loose path. If we find a reachable inescapable subgraph that does not include all the goal path vertices, the reconfiguration problem has no solution. If we find a reachable inescapable subgraph that does include all the goal path vertices, we can solve the reconfiguration problem by applying Lemma 2.

If we find a reachable loose path R for the given instance (G, P, Q), we apply Lemma 5 a second time, to the equivalent reversed instance (G, Q, P). If we find a reachable inescapable subgraph that does not include all the vertices of the original start path P, the reconfiguration problem has no solution. If we find a reachable inescapable subgraph that does include all the vertices of P, we can solve the reconfiguration problem by applying Lemma 2.

If we find a second reachable loose path R', one that (by time-reversal symmetry) can reach the goal configuration, then the original reconfiguration problem has a positive solution. For, in this case, we can reconfigure P to a path that lies along R, then (by Lemma 4) to a path that lies along R', then (by the reverse of the reconfiguration sequence found by the second instance of Lemma 5) to Q. ◻

We leave as open the question of whether a similar result can be obtained for the optimization problem.

4 Tree-like graphs

In the full version of this paper we show that several special classes of graphs have polynomial algorithms for path reconfiguration regardless of path length. The prototypical example are the trees, for which the existence of a polynomial time algorithm follows immediately from the fact that any n-vertex tree has $O(n^2)$ distinct paths. In the full version, we refine this idea and provide a linear time algorithm for path reconfiguration in trees.

We also observe that the graphs of bounded circuit rank, and the graphs of bounded feedback vertex number, have polynomial algorithms for path reconfiguration, because in these graphs the size of the state space (the number of distinct paths in the graph) is bounded by a polynomial. For circuit rank the exponent of the polynomial is a constant, and we obtain a fixed-parameter tractable algorithm. For feedback vertex number, the exponent depends on the feedback vertex number. We defer the details to the full version of the paper.

5 Hardness

In the full version of this paper we describe a reduction from nondeterministic constraint logic showing that path reconfiguration (with unbounded path length) is PSPACE-complete even on graphs of bounded bandwidth. This result rules out the possibility (unless $\mathsf{P} = \mathsf{PSPACE}$) that our results on tree-like graph classes from Section 4 can be extended to another tree-like class of graphs, the graphs of bounded treewidth.

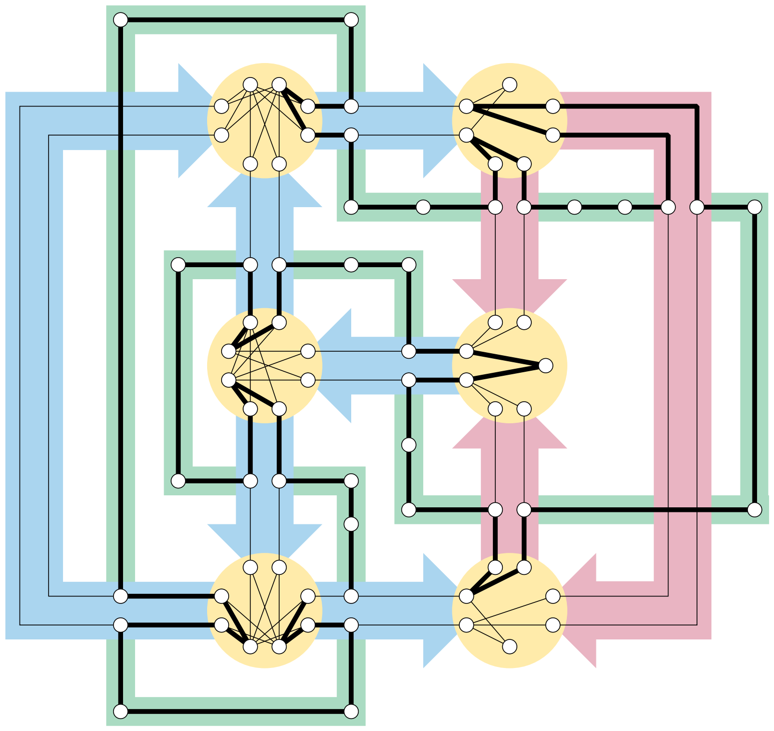

Fig. 6. Reduction from nondeterministic constraint logic to path reconfiguration. The underlying constraint logic instance has six vertices (yellow shaded circles) and nine edges (thick red and blue shaded arrows). Within each of the shaded circles is a vertex gadget of our reduction, and within each thick shaded arrow is an edge gadget of our reduction. The thin green shaded regions contain connection gadgets of our reduction, which the path that is undergoing reconfiguration uses to pass from one edge or vertex gadget to another. The heavy black edges depict one possible state of the path to be reconfigured.

Theorem 2. *The path reconfiguration decision problem is* PSPACE*-complete, even for graphs of bounded bandwidth.*

An example of our reduction is depicted in Figure 6.

References

1. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C.H., Sideri, M., Uehara, R., Uno, Y.: On the complexity of reconfiguration problems. Theoretical Computer Science **412** (2011) 1054–1065
2. Mouawad, A.E., Nishimura, N., Raman, V., Simjour, N., Suzuki, A.: On the parameterized complexity of reconfiguration problems. Algorithmica **78** (2017) 274–297

3. Kamiński, M., Medvedev, P., Milanič, M.: Shortest paths between shortest paths. Theoretical Computer Science **412** (2011) 5205–5210
4. Bonsma, P.: The complexity of rerouting shortest paths. Theoretical Computer Science **510** (2013) 1–12
5. Hanaka, T., Ito, T., Mizuta, H., Moore, B., Nishimura, N., Subramanya, V., Suzuki, A., Vaidyanathan, K.: Reconfiguring spanning and induced subgraphs. In Wang, L., Zhu, D., eds.: Computing and Combinatorics: 24th International Conference, COCOON 2018, Qing Dao, China, July 2–4, 2018, Proceedings. Volume 10976 of Lecture Notes in Computer Science., Springer (2018) 428–440
6. De Biasi, M., Ophelders, T.: The complexity of Snake. In Demaine, E.D., Grandoni, F., eds.: 8th International Conference on Fun with Algorithms, FUN 2016, June 8–10, 2016, La Maddalena, Italy. Volume 49 of Leibniz International Proceedings in Informatics (LIPIcs)., Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2016) 11:1–11:13
7. Gupta, S., Sa'ar, G., Zehavi, M.: The parameterized complexity of motion planning for snake-like robots. Electronic preprint arxiv:1903.02445 (2019)
8. Nešetřil, J., Ossona de Mendez, P.: Chapter 6. Bounded height trees and tree-depth. In: Sparsity: Graphs, Structures, and Algorithms. Volume 28 of Algorithms and Combinatorics. Springer (2012) 115–144
9. Wrochna, M.: Reconfiguration in bounded bandwidth and tree-depth. Journal of Computer and System Sciences **93** (2018) 1–10
10. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (STOC '74) (1974) 172–184
11. Bodlaender, H.L.: On linear time minor tests with depth-first search. Journal of Algorithms **14** (1993) 1–23
12. Fellows, M.R., Langston, M.A.: On search, decision, and the efficiency of polynomial-time algorithms. Journal of Computer and System Sciences **49** (1994) 769–779
13. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM **42** (1995) 844–856