Rerouting shortest paths in planar graphs

Paul Bonsma

Computer Science Department, RWTH Aachen University, Germany.
bonsma@informatik.rwth-aachen.de

Abstract

A rerouting sequence is a sequence of shortest st-paths such that consecutive paths differ in one vertex. We study the Shortest Path Rerouting Problem, which asks, given two shortest st-paths P and Q in a graph G, whether a rerouting sequence exists from P to Q. This problem is PSPACE-hard in general, but we show that it can be solved in polynomial time if G is planar. To this end, we introduce a dynamic programming method for reconfiguration problems.

1998 ACM Subject Classification G.2.2 (Graph algorithms)

Keywords and phrases shortest path, rerouting, reconfiguration problem, planar graph, polynomial time, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.337

1 Introduction

In this paper, we study the *Shortest Path Rerouting (SPR)* Problem, as introduced by Kamiński et al. [14]. An instance of this problem consists of a graph G with unit edge lengths, two vertices $s, t \in V(G)$, and two shortest st-paths P and Q. Shortest st-paths are *adjacent* if they differ in one vertex. The question is whether there exists a *rerouting sequence* from P to Q, which is a sequence of shortest st-paths $Q_0, \ldots, Q_k$ with $Q_0 = P$, $Q_k = Q$, such that consecutive paths are adjacent. This question may arise for instance when a commodity is routed in a network along a shortest path, and a different shortest path route is desired. However, changing the path can only be done by exchanging one node at a time, and transfer should not be interrupted [14]. A different setting where this problem may occur is if a shortest path changes randomly over time, and one wishes to know which paths are reachable from a given starting path.

On the negative side, this problem is known to be PSPACE-complete [2]. Furthermore, Kamiński et al. [14] describe instances where the minimum length of a rerouting sequence is exponential in $n = |V(G)|$. In addition they show that it is strongly NP-hard to decide whether a rerouting sequence of length at most k exists [14]. On the positive side, in [2] it is shown that SPR can be solved in polynomial time in the case where G is claw-free or chordal. In these cases, the related problem of deciding whether there is a rerouting sequence between *every* pair of shortest st-paths is also shown to be solvable in polynomial time, and in the chordal case, a shortest rerouting sequence can be found efficiently.

In this paper, we study the SPR Problem for planar graphs, and prove that this case can also be solved in polynomial time. Questions related to (rerouting) shortest paths occur often in networks, and these are often planar in practice, so this is a relevant graph class for this problem.

Similar questions about the reachability of solutions can be asked for many different combinatorial problems. This only requires defining a (symmetric) adjacency relation between feasible solutions. Ito et al. [12] called such problems *reconfiguration problems*, and initiated a systematic study of them. To be precise, for a reconfiguration problem, it is necessary

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 337–349
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that both adjacency between solutions and the property of being a solution can be tested in polynomial time [12]. Such reconfiguration problems have been studied often in recent literature, for instance based on vertex colorings [3, 5, 6, 7], independent sets [11, 12, 15], satisfying assignments for boolean formulas [10], matchings [12], and more [9, 12, 13, 14]. Usually, the most natural adjacency relation between solutions is considered, e.g. two vertex colorings are considered adjacent in [3, 5, 6, 7] if they differ in one vertex.

One of the motivations for researching reconfiguration problems is to study the structure of the solution space of well-studied combinatorial problems, which can explain the performance of various heuristics [5, 10]. In addition, similar problems have also occurred in practical applications such as stacking problems in storage spaces [17] and train switch-yards (see [16] and references therein).

The main question that has been studied for reconfiguration problems is the complexity of deciding whether there exists a reconfiguration sequence between a given pair of solutions. For most of the aforementioned problems this problem turned out to be PSPACE-complete, although Ito et al. [12] identified a few reconfiguration problems that can be solved in polynomial time, such as matching reconfiguration. Alternatively, one may ask whether there exist reconfiguration sequences between *all* solutions [5, 6, 10], or study the question of how much the solution space needs to be increased such that a reconfiguration sequence becomes possible [4, 12], e.g. allowing to use more colors in vertex colorings used in a reconfiguration sequence [4]. In addition, one may study the problem of finding shortest reconfiguration sequences, or give upper bounds on their length [7, 10, 14].

Various advanced negative results have been proved for reconfiguration problems, such as the first two (independent) PSPACE-hardness results for such problems, on satisfiability reconfiguration [10] and sliding block puzzles [11]. To our knowledge, the other known PSPACE-hardness results on reconfiguration problems have been proved using reductions from these two results. We remark that various PSPACE-complete problems of a similar flavor have been described earlier, such as in the context of local search [18]. An essential difference is however that these are based on asymmetric adjacency relations.

In contrast, it has turned out to be rather challenging to obtain nontrivial positive results for (special cases) of reconfiguration problems. Only very few advanced polynomial time algorithms are known for reconfiguration problems, such as the algorithm by Cereceda et al. [7] on the reconfiguration of vertex colorings using three colors, and the result from Ito et al. [12] on matching reconfiguration. A reason for this lack may be that no general algorithmic techniques are known for reconfiguration problems. Introducing such techniques, and further showing that advanced positive results are possible for reconfiguration problems, are important motivations for the research presented in this paper.

We now give an outline of our results, an informal introduction to the new techniques and ideas, and sketch some obstacles for SPR in planar graphs. Detailed definitions will be given afterwards in Section 2. Firstly, in polynomial time we can delete all vertices and edges of G that do not lie on any shortest st-path, without affecting the answer. So we may assume that all vertices and edges of G lie on shortest st-paths. The problem is then straightforward in the case where the given graph G is st-planar, i.e. has a (planar) embedding where the end vertices s and t are incident with the same face, say the infinite face. The symmetric difference $\mathcal{C} := E(P)\Delta E(Q)$ of the edge sets of the two given shortest st-paths P and Q gives a set of cycles, with disjoint insides (an embedded cycle divides the plane into two regions; the finite region is called the *inside*). One can easily verify that a rerouting sequence from P to Q exists if and only if all faces inside these cycles $\mathcal{C}$ have length 4. See Figure 1(a) for an example. If G is not st-planar, there are still cases where such

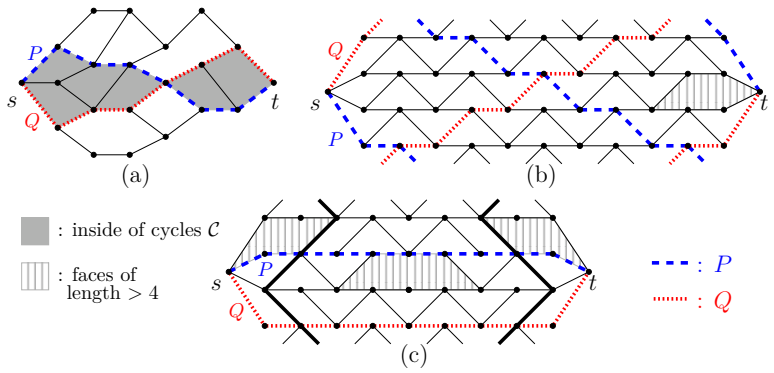

Figure 1 Three examples of planar graphs where a rerouting sequence from P to Q exists. In our figures, half edges leaving the top of the figure continue on the bottom.

a topological viewpoint works well: Figure 1(b) shows an example where every rerouting sequence $Q_0, \ldots, Q_k$ has the property that for any pair of consecutive shortest paths Q_i and Q_{i+1}, the symmetric difference of the edge sets $E(Q_i)\Delta E(Q_{i+1})$ gives a facial 4-cycle. (For readability, some edges are shown as pairs of half edges in our figures; edges leaving the top of the figure continue on the bottom.) Sloppily speaking, rerouting sequences of this type are called *topological*. Even though it is not clear in advance which region of the plane should have only faces of length 4, using some topological intuition it can be verified that also in this example a rerouting sequence from P to Q exists (the unique face of length greater than 4 is shaded). However, not all rerouting sequences in planar graphs are topological. Figure 1(c) shows an example where there exists no topological rerouting sequence from P to Q (consider the shaded faces). The method given in Section 5 will show that nevertheless, there exists a rerouting sequence from P to Q. The reason is that the two non-facial 4-cycles shown in bold can also be used for 'non-topological rerouting steps'. Cycles of this type will be called *switches*.

In Section 4, we will first give an algorithm for the problem of finding topological rerouting sequences. Next, in Section 5, switches are addressed, and our main algorithm for the SPR Problem is presented, which reduces the problem to a polynomial number of instances of the topological version of the problem. This reduction actually requires answering a slightly more general question: instead of asking for a topological rerouting sequence between two given shortest st-paths P and Q, we ask for a topological rerouting sequence from P to some shortest st-path that contains a given set of vertices. This problem is called the *Topological SPR Problem.*

The proofs in Section 5 demonstrate the following simple but powerful principle for reconfiguration problems: instead of searching for a direct reconfiguration sequence between two solutions, it is often easier to identify 'central solutions', and search for two reconfiguration sequences to a common central solution. In our case, central solutions turn out to be paths that contain many switch vertices.

To solve the Topological SPR Problem, we will not further pursue the topological ideas sketched above, but instead solve a more general problem, and develop a dynamic programming method for reconfiguration problems. This is done in Section 3, where an algorithm for the *Restricted SPR Problem* is given. In this generalization of SPR, auxiliary edges are added *(layer edges)*, to encode which rerouting steps are allowed. Our dynamic programming algorithm for Restricted SPR returns the correct answer for all instances, and does not even require the graph G to be planar, but it may require exponential time in some cases.

In Section 4, we show however that this algorithm can be used to solve the Topological SPR Problem in polynomial time. This is not the only purpose of the algorithm however; in Section 3 we also mention other nontrivial (and nonplanar) cases that it solves efficiently. The new algorithmic techniques are discussed in a broader context in Section 6, and open questions are presented. In Section 2, we first give detailed definitions. Because of space constraints, most proofs or proof details are omitted from this extended abstract; they can be found in the full version of the paper.

2 Preliminaries

For graph theoretical notions not defined here (in detail), and background on results mentioned in this section, we refer to [8]. By $N(v)$ we denote the neighborhood of a vertex v. A *walk* of length k from v_0 to v_k in a graph G is a vertex sequence $v_0, \ldots, v_k$, such that for all $i \in \{0, \ldots, k-1\}$, $v_i v_{i+1} \in E(G)$. It is a *path* if all vertices are distinct. It is a *cycle* if $k \geq 3$, $v_0 = v_k$, and $v_0, \ldots, v_{k-1}$ is a path. The corresponding subgraphs will also be called paths and cycles, respectively. A *plane graph* is a graph together with an embedding in the plane (without crossing edges). For planar graphs, an embedding can be found in polynomial time, so it suffices to prove our results for plane graphs. If a plane graph is 2-connected, then the boundary of every face is a cycle, which is called a *facial cycle*. Cycles in plane graphs correspond to simple closed curves, which divide the plane into two regions. For a cycle C in a plane graph G and vertices $s, t \in V(G)$, we say C *separates* s *from* t if s and t lie in different regions of the curve given by C (and thus $s, t \notin V(C)$). Instead of $S \cup \{x\}$ and $S \backslash \{x\}$, we write $S + x$ and $S - x$, respectively.

Throughout this paper, s and t denote two (distinct) vertices of an unweighted, undirected, simple, finite graph G, and we will consider shortest st-paths in G. Let d denote the distance from s to t in G. For $i \in \{0, \ldots, d\}$, we define $L_i \subseteq V(G)$ to be the set of vertices that lie on a shortest st-path, at distance i from s. The vertex set L_i is also called a *layer*.

Observe that a shortest st-path contains exactly one vertex of every layer, and that a shortest path is uniquely determined by its vertex set. Therefore, we will denote shortest st-paths simply by their *vertex set*. A vertex set $Q \subseteq V(G)$ such that there exists a shortest st-path P with $Q \subseteq P$ is called a *shortest st-subpath*. Note that it is not required that Q is actually a path; the vertices of Q are not necessarily from consecutive layers. Since we are only concerned with shortest paths in G between two given terminals s and t, we will call these *S-paths* for short. Shortest st-subpaths will be called *S-subpaths*. When considering reduced instances, defined by the subgraph induced by all shortest paths between two vertices x and y, these definitions refer to x and y. Given an S-path P, a rerouting step consists of replacing a vertex a by another vertex b in the same layer, such that the resulting set is again an S-path. To be precise, let $x, a, y \in P$ such that $x \in L_{i-1}$, $a \in L_i$ and $y \in L_{i+1}$. For any $b \in L_i - a$ with $\{x, y\} \subseteq N(b)$, the *rerouting step* $x, a, y \to x, b, y$ may be applied, which yields the S-path $Q = P - a + b$.

Let G be a graph and $s, t \in V(G)$. The *rerouting graph* $\mathrm{SP}(G, s, t)$ has as set of vertices all S-paths in G. Two paths are adjacent if they differ in exactly one vertex. To distinguish vertices of $\mathrm{SP}(G, s, t)$ from vertices of G, the former will be called *nodes*. Subsets of $V(\mathrm{SP}(G, s, t))$ will be called sets of S-paths or sets of nodes, depending on the context. In order to prove our results, we need to consider two additional variants of the SPR problem, which are defined by considering different adjacency relations. Call a rerouting step $x, a, y \to x, b, y$ a *restricted rerouting step* if $ab \in E(G)$. In the *restricted rerouting graph* $\mathrm{SP}^R(G, s, t)$, two S-paths P and Q are adjacent if Q can be obtained from P using a restricted

rerouting step. Edges $ab \in E(G)$ with $a, b \in L_i$ for some i are called *layer edges*.

In the case that G is a plane graph, we can define a third type of rerouting graph. A sequence of four vertices x, a, b, y is called a *switch* if x, a, y, b, x is a cycle that separates s from t, and for some i, $x \in L_i$ and $y \in L_{i+2}$. (Recall that separating s from t means that e.g. s lies 'inside' the cycle, and t 'outside'.) The vertices x and y are called its *(left and right) switch vertices*. Together, they are also called a *switch-pair*. (Note that in the switch notation x, a, b, y, the vertices are ordered non-decreasingly by their layers, but this is not the order on the cycle.) For instance, the graph G_4 shown in Figure 3 contains exactly one switch: $6, 7, 8, 9$. ($6, 8, 7, 9$ is considered to be the same switch.) A rerouting step $x, a, y \to x, b, y$ is called *topological* if x, a, b, y is not a switch. In the *topological rerouting graph* $\mathrm{SP}^T(G, s, t)$, two S-paths P and Q are adjacent if Q can be obtained from P by a topological rerouting step.

Walks in $\mathrm{SP}(G, s, t)$, $\mathrm{SP}^R(G, s, t)$ and $\mathrm{SP}^T(G, s, t)$ are called *rerouting sequences*, *restricted rerouting sequences*, and *topological rerouting sequences*, respectively. Let P be an S-path, and let Q be an S-subpath. We write $P \rightsquigarrow_G Q$ to denote that in G, there exists a rerouting sequence from P to an S-path Q' with $Q \subseteq Q'$. Similarly, the notations $P \rightsquigarrow_G^R Q$ and $P \rightsquigarrow_G^T Q$ are used for the restricted and topological case, respectively. If the graph G in question is clear, the subscript is omitted. If $P \rightsquigarrow Q$, $P \rightsquigarrow^R Q$ or $P \rightsquigarrow^T Q$, we also say that Q is *reachable* from P. We write $P \not\rightsquigarrow Q$ to denote that $P \rightsquigarrow Q$ does not hold.

The *Generalized Shortest Path Rerouting (GSPR) Problem* asks, given a graph G with $s, t \in V(G)$, an S-path P and an S-subpath Q, whether $P \rightsquigarrow_G Q$. Similarly, for the *Restricted SPR (RSPR) Problem* and *Topological SPR (TSPR) Problem*, it should be decided whether $P \rightsquigarrow_G^R Q$ and $P \rightsquigarrow_G^T Q$, respectively.

Since Q may be an *S-subpath* (in all of these problems), GSPR is a generalization of the SPR Problem (defined in Section 1). The RSPR Problem in turn generalizes the GSPR Problem: a GSPR instance G, P, Q can be transformed to an equivalent RSPR instance by adding edges between every pair of vertices in the same layer. (This may however destroy planarity.)

All algorithmic results and reductions presented in this paper for deciding $P \rightsquigarrow Q$, $P \rightsquigarrow^T Q$ or $P \rightsquigarrow^R Q$ are *constructive*, in the following sense: if $P \rightsquigarrow Q$ for an S-subpath Q, then in addition our algorithms construct an S-path Q' with $P \rightsquigarrow Q'$ and $Q \subseteq Q'$. (Analogously for the topological and restricted case.) For brevity, this is not stated in every lemma and theorem, but this fact is essential for the proofs in Section 5.

For ease of notation, we will assume throughout the paper that in the given graph G, *every vertex lies on an S-path*. Since vertices that do not lie on an S-path are irrelevant for all problems considered here, they may be deleted in advance, which can be done in polynomial time.

3 A Dynamic Programming Algorithm for RSPR

Let P be an S-path in G of length d, and let Q be an S-subpath. We want to decide whether $P \rightsquigarrow_G^R Q$. For $i = 0, \ldots, d-1$, we define the graph G_i as follows: G_i is obtained from G by first removing all vertices in layers $L_{i+1}, \ldots, L_{d-1}$, and then adding edges from t to all vertices in L_i. Using the assumption that every vertex of G lies on an S-path, in particular those in L_{d-1}, we see that $G_{d-1} = G$. By P^i and Q^i we denote $P \cap V(G_i)$ and $Q \cap V(G_i)$, which clearly are again an S-path and an S-subpath in G_i. Figure 2(a) shows an example of G_3, P^3 and Q^3, for the G, P and Q that are shown in Figure 5(a), although in Figure 5(a), we have omitted the layer edges.

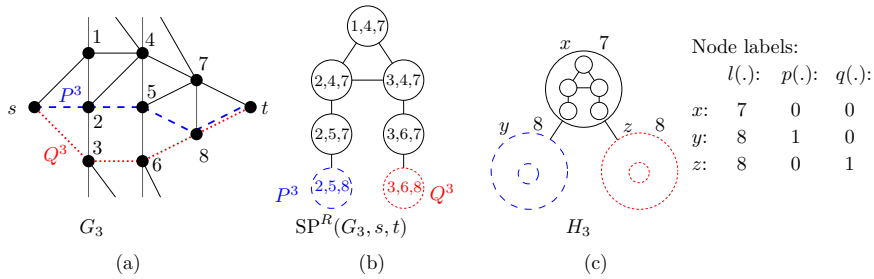

Figure 2 G_3, its restricted rerouting graph $\mathrm{SP}^R(G_3, s, t)$, and a contraction H_3 of it (with nodes x, y, z), which is the encoding of (G_3, P, Q). In the nodes of $\mathrm{SP}^R(G_3, s, t)$, the vertices of the corresponding paths are shown, except s and t. In the nodes of H_3, the corresponding contracted subgraph of $\mathrm{SP}^R(G_3, s, t)$ is drawn.

The idea is now to compute $\mathrm{SP}^R(G_{i+1}, s, t)$ from $\mathrm{SP}^R(G_i, s, t)$, for $i = 0, \ldots, d-2$. In the end, this will yield $\mathrm{SP}^R(G_{d-1}, s, t) = \mathrm{SP}^R(G, s, t)$, and we can decide whether in this graph a path from P to Q exists. The problem is of course that the graphs $\mathrm{SP}^R(G_i, s, t)$ are usually exponentially large compared to G. We solve this problem by instead considering a graph H_i that is obtained from a component of $\mathrm{SP}^R(G_i, s, t)$ by contracting connected subgraphs into single nodes, and using node labels to keep track of essential information about the corresponding path sets.

For two S-paths R and R' in G_i, we define $R \sim_i R'$ if and only if there exists a restricted rerouting sequence from R to R' that does not change the vertex in L_i (so $R \cap L_i = R' \cap L_i$). Clearly, $\sim_i$ is an equivalence relation. Furthermore, if S is an equivalence class of $\sim_i$, then S induces a connected subgraph of $\mathrm{SP}^R(G_i, s, t)$. These are exactly the subgraphs of $\mathrm{SP}^R(G_i, s, t)$ that we will contract to obtain H_i. The following definition is illustrated in Figure 2(b) and (c).

▶ Definition 1 (Encoding). Let P be an S-path in G of length d, let Q be an S-subpath in G, and let $i \in \{0, \ldots, d-1\}$. The *encoding* H_i of (G_i, P, Q) is a node-labeled graph that is obtained from $H' = \mathrm{SP}^R(G_i, s, t)$ as follows:

1. Delete every component of H' that does not contain the node P^i.
2. For every equivalence class $S \subseteq V(H')$ of $\sim_i$ that has not been deleted, contract the subgraph $H'[S]$ into a single node x, and define $S_x := S$ (this is a set of S-paths in G_i). Define $l(x)$ to be the vertex in L_i that is part of every path in S_x. Set $p(x) = 1$ if $P^i \in S_x$, and $p(x) = 0$ otherwise. Set $q(x) = 1$ if there exists an S-path $Q' \in S_x$ with $Q^i \subseteq Q'$ and $q(x) = 0$ otherwise.

H_i is now the resulting graph, together with the node labels $l(x)$, $p(x)$ and $q(x)$ for every node $x \in V(H_i)$.

Note that the encoding H_i defined this way is unique (ignoring node names). We remark that the path sets S_x for every node x are not part of the encoding and do not count towards the size; these sets are only used for the correctness proofs below. We first observe that this definition indeed allows us to decide whether $P \rightsquigarrow^R Q$.

▶ Proposition 2. Let H_{d-1} be the encoding of (G_{d-1}, P, Q). Then $P \rightsquigarrow^R Q$ if and only if H_{d-1} contains a node x with $q(x) = 1$.

Next, we study how the encoding H_{i+1} of (G_{i+1}, P, Q) is related to the encoding H_i of (G_i, P, Q). The objective is that we wish to construct H_{i+1} from H_i without considering $\mathrm{SP}^R(G_{i+1}, s, t)$. An example of this construction is shown in Figure 3. For every $v \in L_{i+1}$,

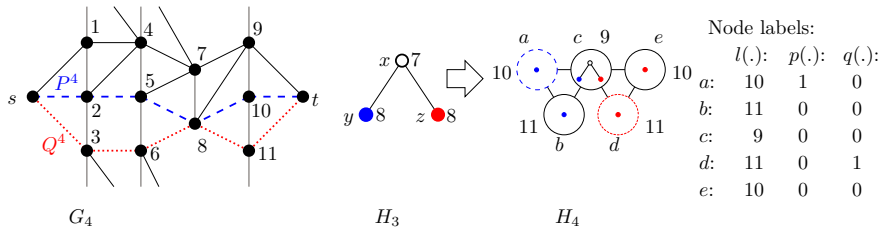

Figure 3 Constructing the encoding H_4 from H_3. In every node a of H_4, the corresponding subgraph C_a of H_3 is drawn. Numbers next to nodes a indicate their label $l(a)$.

let X_v be the set of nodes of H_i that correspond to neighbors of v. Formally, $X_v := \{x \in V(H_i) \mid l(x) \in N(v)\}$. By $H_i^v := H_i[X_v]$ we denote the subgraph of H_i induced by these nodes. This graph may have multiple components, even though H_i is connected. For $v \in L_{i+1}$ and $x \in X_v$, by $S_x \oplus v$ we denote the set obtained by adding v to every path in S_x, so $S_x \oplus v = \cup_{R \in S_x}(R + v)$. Since $l(x) \in N(v)$, this is a set of S-paths in G_{i+1}.

▶ Lemma 3. Let H_i and H_{i+1} be the encodings of (G_i, P, Q) and (G_{i+1}, P, Q), respectively. For any $v \in L_{i+1}$ and component C of H_i^v, $\cup_{x \in V(C)}(S_x \oplus v)$ is a set of S-paths in G_{i+1} that is an equivalence class of $\sim_{i+1}$. In addition, for every $a \in V(H_{i+1})$ with $l(a) = v$, there exists a component C_a of H_i^v such that $S_a = \cup_{x \in V(C_a)}(S_x \oplus v)$.

Lemma 3 shows that for every $a \in V(H_{i+1})$, there exists a corresponding component C of H_i^v, where $v = l(a)$. We denote this component by C_a.

▶ Lemma 4. Let H_{i+1} be the encoding of (G_{i+1}, P, Q). Let $a, b \in V(H_{i+1})$.

(i) $ab \in E(H_{i+1})$ if and only if $l(a)l(b) \in E(G)$ and $V(C_a) \cap V(C_b) \neq \emptyset$.
(ii) $p(a) = 1$ if and only if $l(a) \in P$ and there exists a node $x \in V(C_a)$ with $p(x) = 1$.
(iii) $q(a) = 1$ if and only if $Q \cap L_{i+1} \subseteq \{l(a)\}$ and there exists a node $x \in V(C_a)$ with $q(x) = 1$.

The previous two lemmas give all the information that is necessary to compute H_{i+1} from H_i, including the node labels l, p and q. The essential fact that will guarantee a good complexity bound is that for this computation, knowledge of the path sets S_x for $x \in V(H_i)$ is unnecessary. Together with Proposition 2, this yields a dynamic programming algorithm for deciding $P \rightsquigarrow^R Q$.

▶ Theorem 5. Let G be a graph on n vertices with two vertices $s, t \in V(G)$ at distance d. Let P be an S-path, and let Q be an S-subpath in G. In time polynomial in n and m, it can be decided whether $P \rightsquigarrow^R Q$. Here $m = \max_{i \in \{1,\ldots,d-1\}} |V(H_i)|$, where H_i is the encoding of (G_i, P, Q).

Theorem 5 shows that the RSPR problem can be solved in polynomial time if the size of the encodings H_i remains polynomially bounded. However, since the problem is PSPACE-hard [2], we should not expect this to be true for all graphs. Indeed, there exist examples where the size of the encoding grows exponentially. The example shown in Figure 4 shows that this is even true for the case of planar graphs of maximum degree 6 (or 4, when ignoring layer edges). It can be verified that for $i = 4k - 1$, the encoding H_i is a star with 2^k leaves.

However, there are many nontrivial instances for which this algorithm is polynomial. For instance, we remark (without proof) that this holds for the class of instances described by Kamiński et al. [14], where any rerouting sequence from P to Q has exponential length. (Provided that we swap the choice of s and t in their examples, or in other words, start the

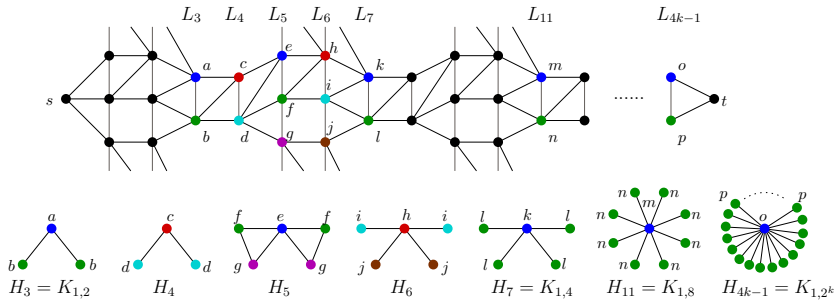

Figure 4 A plane instance of RSPR where the encoding H_i becomes exponentially large. Colors and numbers next to nodes $x \in V(H_i)$ indicate the labels $l(x)$.

computation of encodings from the side of t instead.) In addition, for the following type of low degree instances, the algorithm terminates in polynomial time. The example in Figure 4 shows that the 'degree bounds' in next result are best possible for our algorithm.

▶ Theorem 6. Let G, P, Q be an RSPR instance such that for every i and $v \in L_i$, $|N(v) \cap L_{i-1}| \leq 2$ and $|N(v) \cap L_{i+1}| \leq 2$. Then in polynomial time, it can be decided whether $P \rightsquigarrow^R Q$.

Next, we prove that H_i remains polynomially bounded for instances in a certain standard form, which is closely related to planar graphs.

4 A Polynomial Complexity Bound for TSPR

In this section, we show that if G, P, Q is a (reduced) TSPR instance, then in polynomial time it can be decided whether $P \rightsquigarrow^T Q$. To this end, we define a standard form for RSPR instances, and show for these that the algorithm from Section 3 terminates in polynomial time. Subsequently we show how TSPR instances can be transformed to equivalent RSPR instances in standard form. Figure 5(b) illustrates the following definition.

▶ Definition 7. Consider a graph G and vertices $s, t \in V(G)$ that are part of an RSPR instance. Then G is in *standard form* if the following properties hold:

(i) For every $i \in \{1, \ldots, d-1\}$, $G[L_i]$ has maximum degree 2.
(ii) For every $i \in \{1, \ldots, d-1\}$ and $v \in L_i$, $G[L_{i-1} \cap N(v)]$ is a path.
(iii) For every i and $u, v \in L_i$, if $uv \in E(G)$ then $|N(u) \cap N(v) \cap L_{i-1}| \leq 1$.

If G is in standard form, then with some effort we can show that all encodings H_i are paths, on at most $i \cdot |L_i|$ nodes. Theorem 5 then shows that our dynamic programming algorithm terminates in polynomial time. This gives:

▶ Theorem 8. Let G, P, Q be an RSPR instance in standard form. Then in polynomial time, it can be decided whether $P \rightsquigarrow^R Q$.

The objective is to apply Theorem 8 for TSPR instances G, P, Q. To this end, we will give a polynomial transformation to an equivalent instance G', P, Q of RSPR, and prove that the latter instance is in standard form. In the case of TSPR and GSPR, it will be useful to work with reduced instances. An instance G, P, Q of GSPR or TSPR is *reduced* if:

1. Every vertex and edge of G lies on an S-path,
2. G contains no cut vertices, and

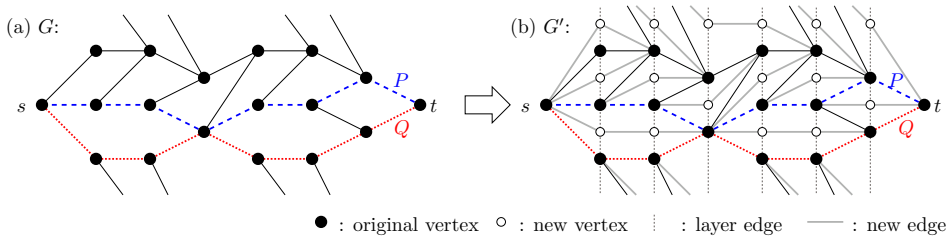

Figure 5 The transformation of a TSPR instance G, P, Q to an equivalent RSPR instance G', P, Q in standard form.

3. G contains no *neighborhood-dominated vertices*, which are vertices z for which there exists a vertex z' with $N(z) \subseteq N(z')$.

We remark that, even though the above definition is useful for both GSPR and TSPR, only in the case of GSPR we can give a polynomial time procedure that can transform every instance to a set of equivalent reduced instances. This procedure is straightforward: we may simply delete all vertices and edges not on S-paths. Next, as long as there exists a neighborhood-dominated vertex z, we may delete z, and replace occurrences of z in P and Q by the corresponding vertex z'. When a cut vertex v is present, the instance basically consists of two independent instances: one induced by all shortest sv-paths, and one induced by all shortest vt-paths.

▶ Theorem 9. Let G, P, Q be a GSPR instance. In polynomial time, a set of reduced GSPR instances can be constructed such that $P \rightsquigarrow_G Q$ if and only if every reduced instance is a YES-instance. If G is plane, all of the reduced instances are plane. The sum of the number of edges of the reduced instances is at most $|E(G)|$.

For $v \in V(G)$, by $\text{dist}_s(v)$ we denote the distance from s to v, so $v \in L_{\text{dist}_s(v)}$. We assume that G, P, Q is reduced, so every vertex and edge of G lies on an S-path. It follows that for every edge uv, it holds that $|\text{dist}_s(u) - \text{dist}_s(v)| = 1$. Furthermore, assuming G is nontrivial, G is 2-connected. So in the case where G is plane, for every face f of G and vertex v incident with f, v has exactly two incident edges uv and vw that are also incident with f. We call v a *local maximum for* f if $\text{dist}_s(v) > \text{dist}_s(u) = \text{dist}_s(w)$, and a *local minimum for* f if $\text{dist}_s(v) < \text{dist}_s(u) = \text{dist}_s(w)$.

▶ Proposition 10. Let G be a 2-connected plane graph in which every vertex and edge lies on a shortest st-path. For every face f of G, there is exactly one local maximum and one local minimum.

Now we can define the transformation from the TSPR instance G to the RSPR instance G'. This transformation is illustrated in Figure 5, and consists of the following two steps.

1. For every face f of G, we do the following. Let u and v be the local minimum and maximum of f. Since G is simple, $\text{dist}_s(u) \leq \text{dist}_s(v) - 2$. Let $\ell = \text{dist}_s(v) - \text{dist}_s(u)$. Add $\ell - 1$ *new vertices* $x_1, \ldots, x_{\ell-1}$, drawn in the face f, and ℓ edges such that $u, x_1, \ldots, x_{\ell-1}, v$ is a path of length ℓ, drawn in face f. Clearly, this preserves planarity. Call the vertices and edges introduced this way *new vertices and edges*. The vertices and edges that were already present in G are called *original vertices and edges*.
2. For every face f in the resulting graph, we do the following. Note that f still has a unique local minimum u and local maximum v. Furthermore, for every edge xy, $|\text{dist}_s(x) - \text{dist}_s(y)| = 1$. It follows that for every $i \in \{\text{dist}_s(u) + 1, \ldots, \text{dist}_s(v) - 1\}$,

there are exactly two vertices a and b incident with f in layer L_i. Between every such pair of vertices a and b, we can add an edge ab, drawn in the face f, without destroying planarity. (Hence the new edges are layer edges.)

Call the resulting plane graph G'. It can be shown that G', P, Q is an RSPR instance in standard form. Since G contains no neighborhood-dominated vertices, for any topological rerouting step $x, a, y \to x, b, y$, it holds that x, a, y, b, x is a facial cycle. So it can be replaced by two restricted rerouting steps $x, a, y \to x, z, y \to x, b, y$ for G', where z is a new vertex. Therefore, $P \rightsquigarrow^T_G Q$ implies $P \rightsquigarrow^R_{G'} Q$. For the converse, it can be shown that rerouting steps in a restricted rerouting sequence for G' can be grouped in pairs $x, a, y \to x, z, y \to x, b, y$ where only z is a new vertex. Hence x, a, y, b, x is a facial cycle of G, and $x, a, y \to x, b, y$ is a topological rerouting step. We conclude that $P \rightsquigarrow^T_G Q$ if and only if $P \rightsquigarrow^R_{G'} Q$. The above transformation is polynomial, so applying Theorem 8 gives:

▶ Theorem 11. Let G, P, Q be a reduced TSPR instance. In polynomial time, it can be decided whether $P \rightsquigarrow^T_G Q$.

5 Reducing Switches, and an Algorithm for SPR

As shown in Figure 4, the presence of switches in a plane graph G may cause our dynamic programming algorithm to take exponential time. However, in this section we show that switches also give a lot of structural information, which can be used to obtain a polynomial time algorithm. If x, a, b, y is a switch in G, then in many cases we can reduce the problem to two smaller subproblems, defined as follows: G_{sy} is the subgraph of G induced by all vertices that lie on a shortest sy-path, and G_{xt} is the subgraph of G induced by all vertices that lie on a shortest xt-path. For an S-subpath Q, we denote $Q_{xt} = Q \cap V(G_{xt})$, and $Q_{sy} = Q \cap V(G_{sy})$. We remark that the rerouting sequences that we consider in G_{sy} (G_{xt}), consist of shortest sy-paths (resp. xt-paths). We are now ready to state the key lemma for reducing the GSPR problem, when switches are present.

▶ Lemma 12. Let G, P, Q be a plane reduced GSPR instance, such that x, y is a switch pair with $\{x, y\} \subseteq P$, and Q is one of the following:

(i) an S-path that contains x and y,
(ii) $Q = \{x', y'\}$ where x', y' is a switch pair, or
(iii) $|Q| = 1$.

Then $P \rightsquigarrow_G Q$ if and only if both $P_{sy} \rightsquigarrow_{G_{sy}} Q_{sy}$ and $P_{xt} \rightsquigarrow_{G_{xt}} Q_{xt}$.

▶ Theorem 13. Let G, P, Q be a plane reduced GSPR instance, where Q is a set containing a switch pair or a single vertex of G. Then in polynomial time it can be decided whether $P \rightsquigarrow Q$.

Proof: First, compute whether $P \rightsquigarrow^T Q$, which can be done in polynomial time (Theorem 11). If yes, then clearly $P \rightsquigarrow Q$ holds as well, and an S-path Q' with $P \rightsquigarrow Q'$ and $Q \subseteq Q'$ can be computed. (Recall that, as discussed in Section 2, all of our algorithms are constructive.) If $P \not\rightsquigarrow^T Q$, then for every switch pair x, y, we compute whether $P \rightsquigarrow^T \{x, y\}$, and if so, compute the corresponding reachable S-path that contains x and y. Since the number of switch pairs in G is polynomial (linear in fact), this can again be done in polynomial time (Theorem 11). If no switch pair is reachable, then note that we may conclude that $P \not\rightsquigarrow Q$.

Now consider a switch pair x, y with $P \rightsquigarrow^T \{x, y\}$. Let P' be the S-path with $P \rightsquigarrow^T P'$ and $\{x, y\} \subseteq P'$ that has been computed. Clearly, $P \rightsquigarrow^T P'$ implies $P \rightsquigarrow P'$ and $P' \rightsquigarrow P$. Therefore, $P \rightsquigarrow Q$ if and only if $P' \rightsquigarrow Q$. By Lemma 12, $P' \rightsquigarrow Q$ if and only if both

$P'_{xt} \rightsquigarrow Q_{xt}$ and $P'_{sy} \rightsquigarrow Q_{sy}$ hold. We decide the latter two properties recursively. This way, we can decide whether $P \rightsquigarrow Q$.

It remains to consider the complexity of this algorithm. We argued that the complexity of the above procedure, not counting the recursive calls, can be bounded by a (monotone increasing) polynomial $\text{poly}(n)$, where $n = |V(G)|$. Recall that d denotes the distance between the end vertices s and t. If there are no switch pairs (which is true in particular when $d \leq 3$), then the entire procedure terminates in time $\text{poly}(n)$.

For $d \geq 3$, we prove by induction over d that the complexity of the algorithm can be bounded by $(2d-5)\cdot\text{poly}(n)$. We have just proved the induction basis ($d = 3$), so now assume $d \geq 4$. In that case, the algorithm may consider a switch pair x, y, and reduce the problem to two instances G_{sy} and G_{xt}. The distance between the end vertices of these instances is d' and $d-d'+2$, respectively, for some $3 \leq d' \leq d-1$. Using the induction assumption and the fact that both G_{sy} and G_{xt} contain at most n vertices, we can bound the total complexity by $(2d' - 5) \cdot \text{poly}(n) + (2(d - d' + 2) - 5) \cdot \text{poly}(n) + \text{poly}(n) = (2d - 5) \cdot \text{poly}(n)$. □

Finally, we are able to prove the main result of this paper.

▶ Theorem 14. Let G be a plane graph, and let P and Q be S-paths in G. In polynomial time, it can be decided whether $P \rightsquigarrow Q$.

Proof sketch: By Theorem 9, we may assume that the instance is reduced. The proof is similar to the proof of Theorem 13. The difference is that now, for every switch pair x, y, we decide whether $P \rightsquigarrow \{x, y\}$ and $Q \rightsquigarrow \{x, y\}$, which can be done in polynomial time (Theorem 13). If the answer differs for P and Q, then we may conclude $P \not\rightsquigarrow Q$. If both $P \rightsquigarrow \{x, y\}$ and $Q \rightsquigarrow \{x, y\}$, then the problem can be reduced using Lemma 12(i), and decided recursively. If $P \not\rightsquigarrow \{x, y\}$ and $Q \not\rightsquigarrow \{x, y\}$ for *every* switch pair x, y, then it suffices to decide whether $P \rightsquigarrow^T Q$, which can be done in polynomial time (Theorem 11). □

6 Discussion

In Section 3, we introduced a dynamic programming method for reconfiguration problems, that can informally be summarized as follows: identify subgraphs G_i, that are separated from the rest of the graph by small separators L_i (in our case, the distance layers). For deciding the reconfiguration problem, detailed information about all solutions is not necessary. So based on how solutions intersect with L_i, one can identify large connected subgraphs (equivalence classes) in the solution graph of G_i, which may be contracted. This method can readily be applied to all kinds of reconfiguration problems, and different sets of separators, in particular small separators given by *tree decompositions* [1]. The challenge is however proving that the resulting encodings stay polynomially bounded. In our case, exponentially large star-like structures (Figure 4) could be avoided by restricting to the topological version of the problem. There may however be different approaches, such as based on reduction rules for the encodings. Exploring this method seems useful firstly for finding other graph classes for which SPR can be solved efficiently, beyond planar graphs and low degree graphs (Theorem 6).

More generally, this dynamic programming method seems useful for answering the following interesting open question: *Are there reconfiguration problems that are PSPACE-hard for graphs of bounded treewidth? Or are there PSPACE-hard reconfiguration problems that can be solved in polynomial time for graphs of treewidth k, for every fixed k?* In contrast to the abundance of positive results on NP-complete problems for bounded treewidth [1], to our

knowledge, not a single (nontrivial) positive or negative result is known for reconfiguration problems on bounded treewidth graphs!

The results presented in this paper can easily be extended to show that if G is planar, shortest (topological) rerouting sequences have polynomial length. Furthermore, a shortest topological rerouting sequence can be found in polynomial time. Whether this also holds for finding a shortest (non-topological) rerouting sequence remains an open question. Recall that if G is not planar, this problem is strongly NP-hard [14].

In addition, the reader may verify that our dynamic programming algorithm can be used to decide efficiently whether $\mathrm{SP}^T(G, s, t)$ is connected. Whether this can also be done efficiently for $\mathrm{SP}(G, s, t)$ is another open question.

References

1. H.L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *ICALP 1988*, volume 317 of *LNCS*, pages 105–118. Springer, 1988.
2. P. Bonsma. The complexity of rerouting shortest paths. In *MFCS 2012*, volume 7464 of *LNCS*, pages 222–233. Springer, 2012.
3. P. Bonsma and L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoret. Comput. Sci.*, 410(50):5215–5226, 2009.
4. L. Cereceda. *Mixing graph colourings*. PhD thesis, London School of Economics and Political Science, 2007.
5. L. Cereceda, J. van den Heuvel, and M. Johnson. Connectedness of the graph of vertex-colourings. *Discrete Appl. Math.*, 308(5–6):913–919, 2008.
6. L. Cereceda, J. van den Heuvel, and M. Johnson. Mixing 3-colourings in bipartite graphs. *European J. Combin.*, 30(7):1593–1606, 2009.
7. L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *J. Graph Theory*, 67(1):69–82, 2011.
8. R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fourth edition, 2010.
9. C.E.J. Eggermont and G.J. Woeginger. Motion planning with pulley, rope, and baskets. In *STACS 2012*, volume 14 of *LIPIcs*, pages 374–383, 2012.
10. P. Gopalan, P.G. Kolaitis, E. Maneva, and C.H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6), 2009.
11. R.A. Hearn and E.D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoret. Comput. Sci.*, 343(1–2):72–96, 2005.
12. T. Ito, E.D. Demaine, N.J.A. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoret. Comput. Sci.*, 412(12–14):1054–1065, 2011.
13. T. Ito, M. Kamiński, and E.D. Demaine. Reconfiguration of list edge-colorings in a graph. In *WADS 2009*, volume 5664 of *LNCS*, pages 375–386. Springer, 2009.
14. M. Kamiński, P. Medvedev, and M. Milanič. Shortest paths between shortest paths. *Theoret. Comput. Sci.*, 412(39):5205–5210, 2011.
15. M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoret. Comput. Sci.*, 439:9–15, 2012.
16. F. König and M. Lübbecke. Sorting with complete networks of stacks. In *ISAAC 2008*, volume 5369 of *LNCS*, pages 895–906. Springer, 2008.

17 F. König, M. Lübbecke, R. Möhring, G. Schäfer, and I. Spenke. Solutions to real-world instances of PSPACE-complete stacking. In *ESA 2007*, volume 4698 of *LNCS*, pages 729–740. Springer, 2007.

18 C.H. Papadimitriou, A.A. Schäffer, and M. Yannakakis. On the complexity of local search. In *STOC 1990*, pages 438–445, New York, 1990. ACM.