Reconfiguring Shortest Paths in Graphs

Kshitij Gajjar · Agastya Vibhuti Jha · Manish Kumar · Abhiruk Lahiri

Abstract

Reconfiguring two shortest paths in a graph means modifying one shortest path to the other by changing one vertex at a time so that all the intermediate paths are also shortest paths. This problem has several natural applications, namely: (a) repaving road networks, (b) rerouting data packets in a synchronous multiprocessor setting, (c) the shipping container stowage problem, and (d) the train marshalling problem. When modelled as graph problems, (a) is the most general case while (b), (c), (d) are restrictions to different graph classes. We show that (a) does not admit polynomial-time algorithms (assuming P \neq NP), even for relaxed variants of the problem (assuming P \neq PSPACE). For (b), (c), (d), we present polynomial-time algorithms to solve the respective problems. We also generalize the problem to when at most k (for a fixed integer k \geq 2) contiguous vertices on a shortest path can be changed at a time.

Keywords

Reconfiguration · Shortest path · PSPACE-complete · Circle graph · Boolean hypercube · Bridged graph · Line graph · Hardness of approximation

An extended abstract of this paper appeared in the proceedings of AAAI 2022 [1]. Kshitij Gajjar’s research was supported by European Union’s Horizon 2020 research and innovation program under Grant agreement No. 682203-ERC-[Inf-Speed-Tradeoff], and an NUS ODPRT Grant WBS No. R-252-000-A94-133. Manish Kumar’s research was supported by the Rita Altura Trust Chair in Computer Science, and by the Lynne and William Franke Center for Computer Science. Abhiruk Lahiri’s research was supported by an Ariel University Post-doctoral fellowship, Israel Science Foundation, Grant number 592/17 and 822/18 and ERC-CZ project LL2005.
1 Introduction

A reconfiguration problem asks questions like: given two different states (configurations) of a system, is it possible to gradually transform (reconfigure) the first configuration to the second by modifying it in a slow, step-by-step manner? Two popular examples of reconfiguration problems are the 15-puzzle [2, 3] and the Rubik’s cube [4, 5]. In both, we want to determine how to reach a “solved” final configuration using a sequence of “moves”, starting from a given initial configuration. Several types of reconfiguration problems have been studied for well-known graph problems, like graph colouring, independent set, domination set, vertex cover, etc. [6–14].

In this paper, we study the reconfiguration problem on shortest paths in graphs, known as the Shortest Path Reconfiguration problem (abbreviated as SPR), introduced by [15]. Let us now define this formally.

Definition 1 Given an undirected, unweighted graph $G$ with a source vertex $s$ and a target vertex $t$, we say that two $s$–$t$ shortest paths $P$ and $Q$ in $G$ are reconfigurable if there is a sequence of $s$–$t$ shortest paths $(P_0, P_1, \ldots, P_{k-1}, P_k)$ where $P_0 = P$ and $P_k = Q$ (for some positive integer $k$) such that $P_i$ and $P_{i+1}$ (for each $i \in \{0, 1, \ldots, k - 1\}$) differ in only one vertex. (See Fig. 1 for an example.)

SPR is the decision problem of checking whether two given shortest paths in a graph are reconfigurable. Additionally, one may ask questions of the following form: If two shortest paths are indeed reconfigurable, is the reconfiguration sequence short enough? If so, is the sequence efficiently computable?
Fig. 1 Reconfiguring a path $P = P_0$ to another path $Q = P_5$ by changing one vertex at a time. Note that all paths in the sequence $(P_0, P_1, P_2, P_3, P_4, P_5)$ are $s$–$t$ shortest paths

SPR has several real-world applications, some of which we describe in Sect. 3. Despite these numerous applications, SPR has not received its fair share of attention from the theoretical standpoint. This is because when research on reconfiguration began almost forty years ago, the primary motivation behind studying the problem was in the context of coordinated motion planning of robots [16]. Large swarms of robots are operated by a central algorithm, which gives specific instructions to each robot so that they can function as a team to solve a given task. Given the initial and final configurations of the robots (a configuration is simply a snapshot of the positions of the robots), the goal of the algorithm is to modify the initial positions of the robots in a sequential, step-by-step manner so that they can reach the final configuration without bumping into each other. (A possible usage of the robots in this setting is to manage a warehouse or inventory.)

Soon thereafter, it was shown that coordinated motion planning of robots is PSPACE-complete [17], implying that there is no polynomial-time algorithm for it unless $P = \text{PSPACE}$. Another closely related problem studied roughly around the same time was known as 2-dimensional planar linkage [18]. Although it was not explicitly stated, it is easy to observe that 2-dimensional planar linkage is essentially a problem about reconfiguring paths of a fixed length on a graph, which is a special case of SPR. For two decades after that, this observation went unexplored and theoretical research in SPR remained dormant. Recently however, there has been a flurry of papers on SPR [15, 19–22].

Bonsma [19] showed that SPR is PSPACE-complete in general. It is also easy to see that SPR on general graphs reduces to SPR on bipartite graphs.
Observation 1  SPR is \(\text{PSPACE}\)-hard even if the input graphs are restricted to be bipartite.

(For completeness, we provide a proof of Observation 1 in Sect. 5.6.) On the positive side, it is known that SPR is solvable in polynomial time for certain graph classes such as planar graphs [20], grid graphs [21], claw-free graphs and chordal graphs [19].

In this paper, we further investigate the complexity of SPR, particularly focusing on graph classes that model real-world problems.

2 Our Contributions and Paper Roadmap

Our contributions are twofold. Firstly, we study SPR for various graph classes. And secondly, we introduce a generalized variant of SPR called \(k\)-SPR. Alongside this, we provide a roadmap of our paper (Table 1).

2.1 SPR: Polynomial-time Algorithms

For circle graphs, permutation graphs and the Boolean hypercube, we provide a complete characterisation of shortest paths and their reconfigurability for SPR. This automatically yields polynomial-time algorithms for them.

For the Boolean hypercube, we show that every shortest path corresponds to a permutation. In fact, the length of the shortest reconfiguration sequence between the two shortest paths is precisely the Kendall’s Tau distance [23] between their respective permutations. The characterisation for circle graphs and permutation graphs is slightly more technically involved. We also solve SPR in polynomial time for a subclass of metric graphs called bridged graphs (more generally, for weakly modular graphs), using a dynamic programming algorithm.\(^{1}\) Finally, for circular-arc graphs and graphs of bounded diameter, we observe that SPR admits simple polynomial-time algorithms.

2.2 \(k\)-SPR: Hardness and Optimization Variants

We introduce a novel generalisation of SPR called \(k\)-SPR, in which we are allowed to change at most \(k\) successive vertices (instead of only one vertex) at a time.

It is known that SPR can be solved in polynomial time for line graphs [19]. We show that \(k\)-SPR is \(\text{PSPACE}\)-complete for line graphs for all integer constants \(k \geq 2\), demonstrating that \(k\)-SPR can be significantly harder than SPR. We also use a “lift-and-project” type proof to show that SPR is \(\text{PSPACE}\)-complete for graph powers by using the \(\text{PSPACE}\)-hardness of \(k\)-SPR.

We observe that for a fixed \(n\), the computational complexity of \(k\)-SPR can decrease as \(k\) increases. More precisely, \(k\)-SPR can be solved in polynomial time when \(k \geq n/2\) (Sect. 6). We also study a few optimisation variants of \(k\)-SPR, and show that there is no polynomial-time algorithm to approximate the “cost” of \(k\)-SPR within a factor.

\(^{1}\) Siggers [24] points out that every SPR instance for bridged graphs and Boolean hypercubes is always a yes-instance, which is implicit in our algorithms.
| Graph class                  | Application (Subsection)          | Result (Subsection)     |
|-----------------------------|-----------------------------------|-------------------------|
| General graphs (k- SPR)     | Repaving Road Networks (3.4)      | PSPACE-complete (4.1)   |
| Permutation graphs          | Train Marshalling (3.2)           | Polynomial-time (5.1)   |
| Circle graphs               | Shipping Container Stowage (3.1)  | Polynomial-time (5.2)   |
| Bridged graphs              | –                                 | Polynomial-time (5.3)   |
| Boolean hypercube           | Rerouting Data Packets (3.3)      | Polynomial-time (5.4)   |
| Circular-arc graphs         | –                                 | Polynomial-time (5.5)   |
| Constant diameter graphs    | –                                 | Polynomial-time (5.6)   |
| Line graphs (k- SPR)        | –                                 | PSPACE-complete (4.1)   |
| Graph powers                | –                                 | PSPACE-complete (4.2)   |
of $O(2^{n^2})$, unless $\textsf{PSPACE} = \textsf{P}$ (Sect. 7). Finally, we examine the gradation of the maximum number of different shortest paths in $n$-vertex graphs as the distance between $s$ and $t$ varies from $O(1)$ to $\Omega(n)$ (Sect. 8).

## 3 Applications

In this section, we study the SPR problem on some restricted graph classes and discuss their usage in practice.

### 3.1 The Shipping Container Stowage Problem

SPR for circle graphs is applicable in maritime transport. Around 80% of all traded goods are transported by sea [26]. Cargo shipping is a billion-dollar industry which leaves a considerable carbon footprint on the environment [27]. Therefore, an efficient process for stowing freight containers on cargo ships is desirable. The process of shifting these containers is an expensive, time-consuming and delicate task. The problem of minimizing the amount of shifting, given a ship’s voyage plan, is known as the container stowage problem. Owing to its importance, this problem has been studied extensively [28–32]. A slight variation of this problem, called the blocks relocation problem has also been studied [33, 34].

One can model the container stowage problem as a graph by representing each container as a vertex, wherein two vertices are adjacent if and only if the loading of one container requires the unloading of the other. These graphs are called overlap graphs. In fact, a graph is an overlap graph if and only if it is a circle graph [25] (Fig. 2). Using this, it was shown that it is $\textsf{NP}$-complete to minimize the amount of unloading/reloading of containers [30, 31]. However, there are two heuristics that give an approximate solution efficiently [28, 34]. One heuristic uses a shortest path-based solution [34], while the other reshuffles the containers in a smart way while limiting the number of possible moves for each container [29].

When the containers are reshuffled at a port, a major operational challenge is to maintain the quality of the solution. Unloading a container (say $C$) at its destination

---

**Fig. 2** An overlap graph (above) for a cargo ship, and its corresponding circle graph (below). Each interval in the overlap graph represents a freight container, its two endpoints being the loading/unloading times for that container on the ship. *(This figure is inspired by a similar figure from [25])"
port requires removing the containers stowed above it (called over-stowed containers). As all these over-stowed containers are adjacent to the vertex \( C \) in the overlap graph, a good strategy is to maintain a shortest path from \( C \) to the vertex that corresponds to the container at the top of \( C \)'s stack at each port. If an extra container is added at some port or an existing container is removed from some port, we should be able to quickly reconfigure the earlier unloading/reloading configuration to a new optimal unloading/reloading configuration.

### 3.2 The Train Marshalling Problem

We solve SPR for circle graphs by solving SPR for a subclass of circle graphs known as permutation graphs and then generalizing our solution to circle graphs. In fact, permutation graphs themselves model a problem that is very much similar to container stowage called the train marshalling problem [35–39].

Both permutation graphs and circle graphs also have applications in memory allocation for system programs [40, 41]. For a comprehensive survey on permutation graphs and circle graphs, see [42, 43].

### 3.3 Rerouting Data Packets

In an efficient synchronous multiprocessing environment, it is widely assumed that there is a common memory and processors having sequential capabilities can access it simultaneously and almost arbitrarily [44]. Such a network of processors is realised by a \( d \)-dimensional Boolean hypercube [45]. The routing of message packets in such a network happens via a greedy scheme which follows shortest paths [46]. The main challenge here is to perform routing in a congestion-free manner, and a lot of research had gone into this [47, 48]. A natural solution is to gradually reroute the packets to a different route [49], which is precisely the SPR problem on the Boolean hypercube.

### 3.4 Repaving Road Networks

\( k \)-SPR has a natural application in restructuring road networks. Suppose you are a city planner and your city’s road network needs to be repaved to better serve the requirements of its residents. For this, you want to change the route between two point locations \( s \) and \( t \) in the city. It is not possible to change the entire route in one go, as laying out new roads takes resources, effort and time. Furthermore, this transition should be smooth. You do not want your ongoing renovation project to cause undue congestion on some roads, leading to a disruption in the overall flow of traffic. In other words, your job is to alter the \( s-t \) route gradually (one road at a time), whilst ensuring that road commuters do not have to undertake a longer route from \( s \) to \( t \) during the process.

A more-or-less similar scenario arises in the case of road accidents [50]. This can sometimes lead to a certain road becoming inoperable, leading to bottleneck situations.
that could increase the travel times of the commuters. In this case, it should be possible
to quickly find a way to reroute the traffic gradually and efficiently.

In SPR, only one vertex can be changed at each reconfiguration step, by definition.
This condition can be sometimes too restrictive for practical purposes. When a graph
is used to model a road network, roads are generally represented by simple induced
paths, and vertices on the path represent various landmarks like bus stops, gas stations,
shops, etc. [51–54].

To model the fact that all these consecutive vertices can be changed in one go, we
introduce the \( k \)-SPR problem, where one can change at most \( k \) (for some fixed positive
integer \( k \)) contiguous vertices at each reconfiguration step. We study the optimization
variant of \( k \)-SPR, where each road has a “cost of construction” associated with it and
the aim is to produce a reconfiguration sequence whose total construction cost is close
to optimal.

We also study \( k \)-SPR for line graphs and graph powers. These graph classes give
us interesting theoretical results that enhance our understanding of SPR. Optimization
variants of other types of reconfiguration problems (e.g., reconfiguring swarm robots)
have also been studied previously [55, 56].

\section*{4 Hardness Results}

\begin{definition} \((k\text{-}SPR)\)
Let \((v_1, v_2, \ldots, v_r)\) be an \(s\text{–}t\) shortest path. Then for each \(1 \leq i < j \leq r\) such that \(j - i < k\), one may replace the subpath \((v_i, v_{i+1}, \ldots, v_j)\) by a
completely new subpath \((u_i, u_{i+1}, \ldots, u_j)\) in a single reconfiguration step of \(k\)-SPR.
\end{definition}

\(k\)-SPR for \(k = 1\) is precisely the SPR problem, and SPR is \textsf{PSPACE}-complete [19].
However, it is important to note that the \textsf{PSPACE}-hardness of SPR does not straigh-
taway imply the \textsf{PSPACE}-hardness of \(k\)-SPR. We show that \(k\)-SPR is \textsf{PSPACE}-hard,
even for a restricted graph class called line graphs.

\subsection*{4.1 Line Graphs}

In this section, we will see that \(k\)-SPR (for \(k \geq 2\)) can be significantly harder than
SPR. In particular, we show that \(k\)-SPR (for all \(k \geq 2\)) is \textsf{PSPACE}-hard for line
graphs. On the other hand, SPR (or \(k\)-SPR for \(k = 1\)) can be solved in polynomial
time for line graphs [19] (in fact, for claw-free graphs, a superclass of line graphs).

\begin{definition}
Given a graph \(G\) on \(m\) edges, the line graph of \(G\), denoted by \(L(G)\), is
an \(m\)-vertex graph where each vertex of \(L(G)\) corresponds to an edge of \(G\), such that
two vertices of \(L(G)\) are adjacent if and only if their corresponding edges in \(G\) share
a vertex (see Fig. 3 for an example).
\end{definition}

\begin{theorem}
\(k\)-SPR is \textsf{PSPACE}-complete for all \(k \geq 2\), even when the input graphs
are restricted to be line graphs.
\end{theorem}

\begin{proof}
Fix an integer \(k \geq 2\). We reduce SPR on general graphs to \(k\)-SPR on line graphs.
Consider an SPR instance \((G, s, t, P, Q)\), where \(P\) and \(Q\) are \(s\text{–}t\) shortest paths in

$G' = L(G_k^*)$ to show PSPACE-hardness of $k$-SPR for $k = 5$. The two $s$–$t$ shortest paths indicated in $G$ (in bold red) differ in only one vertex. Their corresponding $s^*–t^*$ shortest paths (in bold green) differ in 5 vertices in $G'$.

$G$. The goal is to check if $P$ and $Q$ are reconfigurable in $G$. From $(G, s, t, P, Q)$, we construct a $k$-SPR instance $(G', s', t', P', Q')$, where $P'$ and $Q'$ are $s'–t'$ shortest paths in $G'$, such that $P'$ and $Q'$ are $k$-reconfigurable in $G'$ if and only if $P$ and $Q$ are reconfigurable in $G$. Also, $G'$ is a line graph that can be constructed from $G$ in polynomial time in three steps, explained below (see Fig. 4 for an illustration of these steps).

Step (i): Consider the layered graph representation of $G$, with $s$ being the zeroth layer and $t$ being the last layer. This can be done by constructing a BFS tree rooted at $s$. Now replace every “even-odd” edge (i.e., every edge connecting a vertex in layer $i$ to a vertex in layer $i + 1$, for every even $i$) by a path on $k$ vertices between the two endpoints of the edge. Note that if $k = 2$, then this last operation does nothing. Let this new graph be $G_k$, and the new paths corresponding to $P$ and $Q$ in $G_k$ be $P_k$ and $Q_k$, respectively.

Step (ii): Add two vertices $s^*$ and $t^*$ to $G_k$ such that $s^*$ is adjacent only to $s$, and $t^*$ is adjacent only to $t$. Let this new graph be $G_k^*$. The start vertex of $G_k^*$ is $s^*$ and the target vertex of $G_k^*$ is $t^*$. Thus, each $s$–$t$ shortest paths of $G$ corresponds to an $s^*–t^*$ shortest paths of $G_k^*$ whose first edge is always $(s^*, s)$ and last edge is always $(t, t^*)$.

Step (iii): Let $G' = L(G_k^*)$. Since $G'$ is the line graph of $G_k^*$, each vertex of $G'$ is labelled by two vertices of $G_k^*$. That is, a vertex $xy$ in $G'$ (where $x$ and $y$ are two adjacent vertices of $G_k^*$) corresponds to an edge $(x, y)$ in $G_k^*$. The vertex $s^*s$ is our start vertex $s'$ and the vertex $tt^*$ is our target vertex $t'$. 
This completes our construction of $G'$. The paths $P'$ and $Q'$ in $G'$ have their first vertex as $s^*s$, and their last vertex as $tt^*$. Their remaining vertices are the edges on the paths $P_k$ and $Q_k$, respectively. Given the fact that $P$ and $Q$ are $s$–$t$ shortest paths in $G$, it is easy to check that $P'$ and $Q'$ are $s'$–$t'$ shortest paths in $G'$. This completes the definition of the $k$-SPR instance $(G', s', t', P', Q')$. We make the following claim, whose proof will complete our proof of Lemma 1.

**Claim 1** $(G, s, t, P, Q)$ is a yes-instance of SPR $\iff (G', s', t', P', Q')$ is a yes-instance of $k$-SPR.

$\Rightarrow$ direction: Every reconfiguration step in $G$ changes some vertex $u_i$ in layer $i$ to a vertex $v_i$ in the same layer, where $(u_{i-1}, u_i, u_{i+1}, v_i)$ is a 4-cycle in $G$. Note that $u_i$ can never be $s$ or $t$, so it cannot be present in the zeroth or last layer of $G$. Thus, the graph $G_k^*$ contains a vertex $u_{i-2}$ (possibly $s^*$) and a vertex $u_{i+2}$ (possibly $t^*$). Both $u_{i-2}u_{i-1}$ and $u_{i+1}u_{i+2}$ are vertices in the line graph $G' = L(G_k^*)$. Among the two edges $(u_{i-1}, u_i)$ and $(u_i, u_{i+1})$ in $G$, one is retained as an edge in $G_k$ and one is converted to a path on $k$ vertices in $G_k$ (depending on whether $i$ is odd or even). The retained edge contributes to a single vertex in the line graph $G'$, and the path on $k$ vertices (or $k-1$ edges) contributes $k-1$ vertices to the line graph $G'$.

These $k$ vertices are reconfigured to another set of $k$ vertices on the path $Q'$ in $G'$.

$\Leftarrow$ direction: Consider a reconfiguration step in $G'$ which replaces a subpath of $j$ (where $j \leq k$) vertices on a shortest $s'$–$t'$ path by another subpath of $j$ vertices. Since $G'$ is the line graph of $G_k^*$, these $j$ vertices of $G'$ can be mapped back to a subpath of $j$ edges in $G_k^*$ (i.e., a subpath of $j+1$ vertices in $G_k^*$). Let $x$ and $y$ be the first and last vertices of the subpath comprised by these $j+1$ vertices in $G_k^*$. It is easy to see that neither $x$ nor $y$ is changed by mapping the reconfiguration step in $G'$ back to a reconfiguration step in $G_k^*$. Note that $x$ is adjacent to at least two vertices in the next layer in $G_k^*$ (thus $x \neq s^*$ and so $x \in G_k$) and $y$ is adjacent to at least two vertices in the previous layer in $G_k^*$ (thus $y \neq t^*$ and so $y \in G_k$). Therefore, $x$ and $y$ can be mapped back to vertices $\im(x)$ and $\im(y)$ in $G$, because all “new” vertices of $G_k$ are adjacent to only one vertex in the next layer and only one vertex in the previous layer. Finally, if $\im(x)$ is in layer $i$ of $G$ (for some $i$), then $\im(y)$ must be in layer $i+2$ of $G$. This is because if $\im(y)$ lies in a layer before $i+2$ (i.e., $i+1$), then $(\im(x), \im(y))$ would be a multiple edge in $G$, which is a contradiction. And if $\im(y)$ lies in a layer after $i+2$, then $x$ and $y$ would have more than $k$ edges between them in $G_k$. This is also a contradiction, since $j \leq k$.

### 4.2 Graph Powers

In this section, we will show that it is possible to use the PSPACE-hardness of $k$-SPR to prove PSPACE-hardness of SPR for some graph classes, namely graph powers.

**Definition 4** The $k^{th}$ power of a graph $G$, denoted by $G^k$, is obtained by making all those pairs of vertices $(u, v)$ adjacent whose distance in $G$ is at most $k$. In other words,

$$(u, v) \in G^k \iff d_G(u, v) \leq k.$$
Theorem 2   

**Proof**   

Our proof technique is as follows. Let $G^k$ be the $k$th graph power of $G$. We use the PSPACE-hardness of $(2k - 1)$-SPR for $G$ to show the PSPACE-hardness of SPR (or 1-SPR) for $G^k$. 

Let $P_1$, $P_2$ be two reconfigurable $s$–$t$ shortest paths in $G^k$. Consider the reconfiguration sequence. At each step of the reconfiguration sequence, we replace a vertex $v$ with $v'$ where both of them have edges to $u$ and $w$ that belong to intermediate $s$–$t$ shortest path. We can construct a reconfiguration sequence for two $s$–$t$ shortest paths $P'_1$, $P'_2$ in $G$, where $V(P_1) \subseteq V(P'_1)$ and $V(P_2) \subseteq V(P'_2)$, as follows:

- If the edges $uv, vw, u'v', v'w \in E(G)$ then reconfiguration step remain unchanged
- If any of the $uv, vw, u'v', v'w \in E(G)$ then consider a shortest path between the two vertices in $G$. Replace all of them in a single step.

Clearly, following the above steps it is possible to $P'_2$ from $P'_1$. The number of vertices we change in the second case is at most $2k - 1$ as an edge in $G^k$ implies that there exists a path of length at most $k$ between them in $G$. If both edges involved in the reconfiguration step from $v$ or $v'$ are in $G$, then we change at most $2k - 1$ vertices in one step. Hence, it is a $(2k - 1)$-SPR.

To prove the other direction, let $P'_1$ and $P'_2$ be two reconfigurable $s$–$t$ shortest paths in $G$. Each reconfiguration steps in $G$ changes at most $(2k - 1)$ contiguous vertices. Fix a reconfiguration step that changes $k'$ contiguous vertices between two vertices $u$ and $v$ in $G$. Clearly, $k' \leq 2k - 1$. Consider the $[k'/2]^{th}$ vertex after $u$ on $P_1$, and call it $x$. Similarly consider the $[k'/2]^{th}$ vertex after $u$ on $P_2$, and call it $y$. Note that $(u, x)$, $(x, v)$ are edges in $G^k$, as are $(u, y)$ and $(y, v)$. Thus, they can trivially be reconfigured in $G^k$. This completes the proof.

5 Algorithms

In this section, we present polynomial-time algorithms for SPR on circle graphs, bridged graphs, Boolean hypercubes, and some other graph classes.

5.1 Permutation Graphs

Permutation graphs are a subclass of circle graphs, and therefore the algorithm for circle graphs that we present in the next section (Sect. 5.2) also holds for permutation graphs. However, it is instructive to study the polynomial-time algorithm for permutation graphs first, and then generalise it to circle graphs, as the circle graphs algorithm borrows several ideas from the permutation graphs algorithm.

**Definition 5** A graph $G$ on $n$ vertices $\{1, 2, \ldots, n\}$ is called permutation graph if there exists a permutation $\sigma = (\sigma(1)\sigma(2) \cdots \sigma(n)) \in S_n$ such that for every $1 \leq i < j \leq n$, we have $(i, j) \in E(G)$ if and only if $\sigma(i) > \sigma(j)$.

Given a graph, its permutation representation can be constructed in linear time, if it exists [42]. Let $G$ be a permutation graph on $n$ vertices with two special vertices $s$
and $t$. We delete all the edges of $G$ which do not lie on an $s$–$t$ shortest path, and label the edges of $G$ as L-type (L stands for left) or R-type (R stands for right) as follows.

**Definition 6** Let $e$ be an edge of an $s$–$t$ shortest path $P = (s = v_1, v_2, \ldots, v_l = t)$ in $G$. Then $e = (v_i, v_{i+1})$ for some $1 \leq i \leq l-1$. We say that $e$ is of L-type if $v_{i+1} < v_i$, and of R-type if $v_i < v_{i+1}$. (See Fig. 5 for an example.)

Throughout this proof, we will assume that $d(s, t) \geq 3$ (that is, $l \geq 4$) and that $t$ is to the left of $s$, as shown in Fig. 6. This means that for every $3 \leq i \leq l$,

$$(v_i, s) \notin E(G), \ v_i < s \text{ and } \sigma(v_i) < \sigma(s). \quad (1)$$

**Lemma 1** Let $P = (s = v_1, v_2, \ldots, v_l = t)$ be a shortest path in $G$. Then for every $2 \leq i \leq l-1$, the edges $(v_{i-1}, v_i)$ and $(v_i, v_{i+1})$ are of different types.

**Proof** We will prove this lemma by contradiction. Suppose $(v_{i-1}, v_i)$ and $(v_i, v_{i+1})$ are of the same type (say L-type). Then it is easy to see the following.

$$v_{i+1} < v_i < v_{i-1};$$

$$\sigma(v_{i-1}) < \sigma(v_i) < \sigma(v_{i+1}).$$

These imply that $(v_{i-1}, v_{i+1})$ is also an edge, which is impossible in a shortest path.

**Lemma 2** Two shortest paths $P_1$ and $P_2$ can be reconfigured in $G$ if and only if the first edge on both the paths is of the same type. Further, the reconfiguration sequence can be obtained in linear time.

**Proof** Let the paths be

$$P_1 = (v_1, v_2, v_3, \ldots, v_{l-1}, v_l) \text{ and } P_2 = (u_1, u_2, u_3, \ldots, u_{l-1}, u_l),$$

where $v_1 = u_1 = s$ and $v_l = u_l = t$.

$\Leftarrow$ direction: We will show that if $(s, v_2)$ and $(s, u_2)$ are of the same type (say L-type), then $P_1$ and $P_2$ can be reconfigured. Our proof is by induction on $l$ (note that
Algorithmica (2024) 86:3309–3338  3321

Fig. 6  $s$ and $t$ are in black, vertices of $P_1$ are in red and vertices of $P_2$ are in blue (Color figure online)

$\ell = d(s, t) + 1$). For $\ell = 3$, this is trivial (the paths are $(s, v_2, t)$ and $(s, u_2, t)$). Now assume that every pair of shortest paths in permutation graphs with $l - 1$ vertices each, both starting with an L-type edge, is reconfigurable.

We will show that $P_1$ and $P_2$, two paths with $l$ vertices each, both starting with an L-type edge, are always reconfigurable. If $v_2 = u_2$, then let $s' = v_2 = u_2$, and the $s'$-$t$ subpaths of $P_1$ and $P_2$ have $l - 1$ vertices each, which can be reconfigured by the induction hypothesis, and we are done. Now, if $v_2 \neq u_2$, assume that $v_2 < u_2$ (the proof for $u_2 < v_2$ is similar). It is helpful to follow Fig. 6 while reading the rest of this proof. Since both $(s, v_2)$ and $(s, u_2)$ are L-type edges, this means that both $(v_2, v_3)$ and $(u_2, u_3)$ are R-type edges, by Lemma 1. We have the following.

\[
\begin{align*}
\text{The first two lines imply that } v_2 < u_3 \text{ and the last two lines imply that } \sigma(u_3) < \sigma(v_2). \\
&v_2 < u_2 < s; \\
&u_2 < u_3; \\
&u_3 < s, \sigma(u_3) < \sigma(s); \\
&\sigma(s) < \sigma(v_2).
\end{align*}
\]

Thus $(v_2, u_3)$ is an edge in $G$. We can therefore reconfigure the path $P_2$ by replacing $u_2$ with $v_2$, obtaining $P'_2 = (s, v_2, u_3, \ldots, t)$. Now, setting $s' = v_2$ in both $P_1$ and $P'_2$, we obtain two $s'$-$t$ paths $(s', v_3, \ldots, t)$ and $(s', u_3, \ldots, t)$, each with $l - 1$ vertices. By the induction hypothesis, these can be reconfigured. Note that our proof also implies that the reconfiguration sequence can be obtained in linear time.

\begin{itemize}
\item[$\Rightarrow$] direction: We will show that if $P_1$ can be reconfigured to $P_2$, then their first edges are of the same type. For this, we will simply show that a reconfiguration step does not change the type of the first edge.
\item Consider a reconfiguration step in which a vertex $v_i$ of $P_1$ is changed to a vertex $w_i$ ($w_i$ may or may not be $u_i$). If $i > 2$, then this clearly does not change the type of the first edge. If $i = 2$, then the new path is
\[
P'_1 = (s, w_2, v_3, v_4, \ldots, v_{l-1}, u_l).
\]
\end{itemize}

Let the first edge $(s, v_2)$ of $P_1$ be of L-type (the proof for R-type is similar). We will show that the first edge $(s, w_2)$ of $P'_1$ is also of L-type. By Lemma 1, the edges $(v_2, v_3)$ and $(v_3, v_4)$ in $P_1$ are of R-type and L-type, respectively. In $P'_1$, since $(v_3, v_4)$ is of L-type, we have $(w_2, v_3)$ is of R-type and $(s, w_2)$ is of L-type (again by Lemma 1).
5.2 Circle Graphs

We will now show that SPR can be solved in linear time for a superclass of permutation graphs called circle graphs, using much of the same ideas we did for permutation graphs.

**Definition 7** A graph is called a circle graph if its vertices can be represented by the chords of a circle such that two vertices have an edge in the graph if and only if their corresponding chords intersect.

The following well-known fact establishes the connection between circle graphs and permutations graphs.

**Fact 1** ([43]) A graph is a permutation graph if and only if it is a circle graph that admits an equator, i.e., an additional chord that intersects every other chord of the circle graph (Fig. 7).

Given a graph, its circle representation can be constructed in quadratic time, if it exists [42]. Let $G$ be a circle graph and $P_1$ and $P_2$ be two $s$–$t$ shortest paths in $G$. For every vertex $v$ we assign it a label $i$, if it appears on the $i^{th}$ level of the BFS tree rooted at $s$. A chord with label $i$ intersects one or more chords with label $i - 1$ and one or more chords with label $i + 1$ (possibly even other chords at label $i$, but we ignore those for our proof). We orient the chord $i$ from the point of intersection of the $i - 1$ chord on chord $i$ (called the first endpoint) to the point of intersection of the $i + 1$ chord on the chord $i$ (called the second endpoint).

**Lemma 3** Every chord has a unique orientation. In other words, a chord cannot receive two different orientations from two different shortest paths.

**Proof** For the sake of contradiction, assume that there exists a chord $v$ (say, labelled $i$) with two orientations. This means that there exists a pair of chords with labels $(i - 1, i + 1)$ which orient $v$ in one direction, and also another pair of chords with labels $(i - 1, i + 1)$ which orient $v$ in the opposite direction. A little bit of thought reveals that the only way this can happen is if at least one of the following is true.

Fig. 7 A circle graph with an equator (indicated by the dotted line) and its corresponding permutation graph
1. The intersection point of a chord of label $i + 1$ lies between the intersection points of two chords of label $i - 1$.

2. The intersection point of a chord of label $i - 1$ lies between the intersection points of two chords of label $i + 1$.

In the first case (Fig. 8, left), let $u_{i-1}$ and $u'_{i-1}$ be the two chords at level $i - 1$, and $u_{i+1}$ be the chord at level $i + 1$. The chord $u_{i+1}$ divides the circle into two parts. It is easy to see that $u_{i-1}$ and $u'_{i-1}$ lie in opposite parts. Since $s$ does not intersect $u_{i+1}$, it also lies in one of these two parts (say, in the same part as $u_{i-1}$). Now, consider the chords on a shortest path from $s$ to $u'_{i-1}$. Note that at least one of these chords must intersect $u_{i+1}$, implying that $d(s, u_{i+1}) \leq d(s, u'_{i-1})$. This is clearly a contradiction.

In the second case (Fig. 8, right), let $u_{i+1}$ and $u'_{i+1}$ be the two chords at level $i + 1$, and $u_{i-1}$ be the chord at level $i - 1$. The chord $u_{i-1}$ divides the circle into two parts. It is easy to see that $u_{i+1}$ and $u'_{i+1}$ lie in opposite parts. Since $t$ does not intersect $u_{i-1}$, it also lies in one of these two parts (say, in the same part as $u_{i+1}$). Now, consider the chords on a shortest path from $u'_{i+1}$ to $t$. Note that at least one of these chords must intersect $u_{i-1}$, implying that $d(u_{i-1}, t) \leq d(u'_{i+1}, t)$. This is clearly a contradiction.

If the circle graph admits an equator, then we can directly use the algorithm for permutation graphs (Fact 1) from the previous section (Sect. 5.1). Otherwise, we define the equator to be an additional chord intersecting both $s$ and $t$. Given Lemma 3, we will now define a labelling scheme of the chords based on their orientation and also on the positions of their endpoints with respect to the equator. Each chord receives one of four possible labels. (This is similar to the L-type and R-type labels in permutation graphs.)

• TT if both endpoints of the chord are above the equator.
• TB if the first endpoint of the chord is above the equator and the second endpoint is below the equator.
• BT if the first endpoint of the chord is below the equator and the second endpoint is above the equator.
• BB if both endpoints of the chord are below the equator.

Here, T stands for top and B stands for bottom. The label of an $s$–$t$ shortest path is simply a concatenation of the labels of all its vertices from $s$ to $t$. For example,
in Fig. 9, the labels are

\[ P_1 = (sB)(BT)(TT)(TB)(BB)(BT)(Tt) \]
\[ P_2 = (sB)(BT)(TT)(TB)(BB)(BT)(Tt) \]
\[ P_3 = (sB)(BT)(TB)(BT)(TB)(BB)(Bt) \]

We are now set to prove our main theorem.

**Theorem 3** Two s–t shortest paths in a circle graph are reconfigurable if and only if they have the same label. Furthermore, the reconfiguration sequence can be obtained in linear time (if it exists).

**Proof** $\Rightarrow$ direction: We will see that if two s–t shortest paths can be reconfigured, they have the same label. Note that the label of each vertex matches up with the label of the vertex before it and the label of the vertex after it. For example, in $P_3$, the fourth and sixth vertices are (BT) and (BB). The first endpoint of the fifth vertex must match the second endpoint of the fourth vertex, and the second endpoint of the fifth vertex must match the first endpoint of the sixth vertex. So the label of the fifth vertex must be (TB). Thus, a reconfiguration step that changes the fifth vertex of $P_3$ can only change it to another vertex whose label is also (TB). Hence, reconfiguration does not change the label of a path. (In the example, $P_3$ cannot be reconfigured to $P_1$ or $P_2$.)

$\Leftarrow$ direction: We will see that if two s–t shortest paths have the same label, they can be reconfigured. In the example shown, let $v_1$ and $u_1$ be the first vertices of $P_1$ and $P_2$, respectively. Both are labelled (sB). Suppose the second endpoint of the chord $u_1$ lies
before (to the left of) the second endpoint of the chord \( v_1 \) on the bottom of the circle. Then, the chord \( u_2 \) must intersect the chord \( v_1 \). This means the \((s, u_1, u_2)\) subpath can be changed to \((s, v_1, u_2)\) by reconfiguring \( u_1 \) to \( v_1 \). Next, we can show that either the chord \( u_3 \) must intersect the chord \( v_2 \) or the chord \( v_3 \) must intersect the chord \( u_2 \), and similar to the proof of Lemma 2, we will eventually reconfigure \( P_1 \) and \( P_2 \). Also, it is easy to see that the reconfiguration sequence thus obtained is of size \( O(n) \). \( \square \)

5.3 Bridged Graphs

We begin this section with some definitions. Let \( G \) be a graph, and \( u, v \) be two vertices of \( G \). Their interval \( I(u, v) \) is the set of all vertices of \( G \) that lie on at least one shortest \( u-v \) path. More formally,

\[
I(u, v) = \{ w \in V(G) : d(u, w) + d(w, v) = d(u, v) \}.
\]

A subset of vertices \( H \) of \( V(G) \) is called convex if for each pair of vertices \((u, v) \in H \times H\), their interval \( I(u, v) \subseteq H \).

**Definition 8** A graph \( G \) is called a bridged graph if the neighbourhood of every convex set in \( G \) is also convex.

It is known that bridged graphs are precisely the graphs in which all isometric cycles have length three \([57, 58]\). In particular, all chordal graphs are bridged. Bonsma \([19]\) showed that SPR can be solved in polynomial time for chordal graphs. We extend Bonsma’s result to bridged graphs.

Let us now look at some properties of bridged graphs. A graph is called weakly modular if it satisfies the following two conditions \([59, 60]\).

- **The quadrangle condition:** \( \forall u, v, w, z \in V(G) \) with \( k := d(u, v) = d(u, w) \), \( d(u, z) = k + 1 \) and \( vz, wz \in E(G) \), \( \exists x \in V(G) \) such that \( d(u, x) = k - 1 \) and \( xv, xw \in E(G) \).
- **The triangle condition:** \( \forall u, v, w \in V(G) \) with \( k := d(u, v) = d(u, w) \) and \( vw \in E(G) \), \( \exists x \in V(G) \) such that \( d(u, x) = k - 1 \) and \( xv, xw \in E(G) \).

Bridged graphs are weakly modular graphs with no induced cycle of length four or five \([60]\). We essentially present a polynomial-time algorithm for SPR for weakly modular graphs. Our algorithm recursively uses the triangle condition from the above definition. For general graphs, such a recursion would make the running time exponential. We use a suitable data structure to make the running time to polynomial.

Before describing the algorithm and the proof let us define the following notation. Let \( P_{s, w, t} \) denote a shortest path between \( s \) and \( t \) which is going through the vertex \( w \).

**Lemma 4** Algorithm 1 solves SPR on weakly modular graphs in \( O(2^\ell n) \) time, where \( \ell \) is the distance between \( s \) and \( t \).

**Proof** From the triangle condition of weakly modular graphs, we know that for a given \( P_{s, u_\ell, t}, P_{s, v_\ell, t}, s, t \) there exists a vertex \( w \) such that both the edges \( u_\ell w \) and \( v_\ell w \) are preset in the graph. Consider a solution for the SPR. In that solution, we move \( u_\ell \rightarrow v_\ell \)
Algorithm 1 SPR for weakly modular graphs

Input: G, paths \( P_s,u,\ell \) and \( P_s,v,\ell \).

1: if \( w_{\ell-1} \not\in P_s,u,\ell \) then
2: Output \( u_l \rightarrow v_l, \text{ SPR}(G, P_s,w_{\ell-1},v,\ell) \)
3: end if
4: if \( w_{\ell-1} \not\in P_s,v,\ell \) then
5: Output \( \text{ SPR}(G, P_s,u,\ell-1,u,\ell), u_l \rightarrow v_l \)
6: end if
7: if \( w_{\ell-1} \not\in P_s,u,\ell \) then
8: \( \text{ SPR}(G, P_s,u_{\ell-1},u,\ell), P_s,w_{\ell-1},u,\ell \), \( u_l \rightarrow v_l, \text{ SPR}(G, P_s,w_{\ell-1},v,\ell) \)
9: end if

Fig. 10 Boolean hypercube for \( d = 3 \)

at some step. Then, what remains is a solution to SPR on paths whose length is reduced by 1. Depending on the fact whether \( w \) is either \( u_{\ell-1} \) or \( v_{\ell-1} \) or \( w \not\in P_s,u,\ell \) we have three subproblems. That is precisely what Algorithm 1 computes.

At every step searching for a \( w \) requires \( O(n) \) time. The number of the subproblem in the recursion is at most \( 2^\ell \). Hence total running time is \( O(2^\ell n) \). \( \Box \)

The running time of Algorithm 1 is clearly exponential when \( \ell = \Theta(n) \). This can be improved. Consider the following data structure.

Definition 9 Lookup\((u_i, v_i)\)

- Takes input \( u_i, v_i \) from the same BFS layer computed from \( s \).
- Outputs \( w_{i-1} \) such that both \( w_{i-1}u_i \) and \( w_{i-1}v_i \in E(G) \).

We construct Lookup\((u_i, v_i)\) by searching for common parents for every pair of vertices in a BFS layer. Implementing Lookup\((u_i, v_i)\) takes \( O(n^3) \) space. Finding a \( w \) at each step using this data structure requires only a constant amount of time. Finally, the BFS naturally partitions the vertices of \( G \) into layers, reducing the running time to \( O(n^2) \). (This lookup table method is essentially a dynamic program.) We conclude this section with the following theorem and its corollary.

Theorem 4 \( \text{SPR} \) can be solved in \( O(n^2) \) time for weakly modular graphs.

Corollary 1 \( \text{SPR} \) can be solved in \( O(n^2) \) time for bridged graphs.
Fig. 11 There are 6 possible ways to go from $s$ to $t$

5.4 Boolean Hypercubes

Definition 10 A $d$-dimensional Boolean hypercube is a graph with vertex set $\{0, 1\}^d$ in which two vertices are adjacent if and only if their corresponding bit strings differ in exactly one of the $d$ coordinates (Fig. 10).

As an input to the SPR problem, we are given two $s$–$t$ shortest paths $P_1$ and $P_2$ of length $\ell$ each in a $d$-dimensional Boolean hypercube. Let $\oplus$ denote the bit-wise XOR operation and $\text{ones}(\ell)$ denote the positions of the ones in the bit string $\ell$. For example, for $d = 5$,

$$s = 00101$$
$$t = 10011$$
$$s \oplus t = 10110$$
$$\text{ones}(s \oplus t) = \{1, 3, 4\}.$$

Given $\text{ones}(s \oplus t)$ in this example, it is easy to see that every $s$–$t$ shortest path has exactly three edges, one edge each dedicated to changing the bit in the 1st, 3rd, and 4th positions. However, the order in which these changes are made could be different. There are $3! = 6$ ways to do this: $(134)$, $(143)$, $(314)$, $(341)$, $(413)$, $(431)$. In other words, there are six $s$–$t$ shortest paths in this example (see Fig. 11). Therefore, we will represent all $s$–$t$ shortest paths as permutations for the rest of this proof.

Observation 2 Let $s$, $t$ be two vertices of a Boolean hypercube such that $d(s, t) = r$. Then, two permutations $P_1$ and $P_2$ can be reconfigured in a single reconfiguration step if and only if there exists a $j \in [r - 1]$ such that

$$P_1 = (i_1i_2i_3 \ldots i_{j-1}i_ji_{j+1}i_{j+2} \ldots i_{r-1}i_r);$$
$$P_2 = (i_1i_2i_3 \ldots i_{j-1}i_{j+1}i_ji_{j+2} \ldots i_{r-1}i_r).$$

That is, $P_1$ and $P_2$ differ only in the positions $j$ and $j + 1$. 
Proof (Proof sketch) We can only change one vertex at time in SPR, and the permutations simply denote the order in which these changes occur. Therefore, if one shortest path changes \(i_j\) and then \(i_{j+1}\) immediately after that, then the only possible option is to swap the order of these two changes: that is, change \(i_{j+1}\) and then \(i_j\) immediately after that.

\[\square\]

Algorithm 2 SPR for Boolean hypercubes with \(r = d(s, t)\)

**Input:** Permutations \(P_1, P_2 \in S_r\)

1. **while** \(\exists i \in \{1, 2, \ldots, r - 1\}\) such that \(P_2^{-1}(P_1[i]) > P_2^{-1}(P_1[i + 1])\) **do**
2. Swap \(P_1[i]\) and \(P_1[i + 1]\) in \(P_1\)
3. **end while**

Theorem 5 Algorithm 2 reconfigures two given \(s\)–\(t\) shortest paths (or permutations) \(P_1\) and \(P_2\) in a Boolean hypercube in the minimum number of reconfiguration steps.

**Proof** It is easy to see that Algorithm 2 reconfigures \(P_1\) to \(P_2\) by changing one vertex of \(P_1\) at each reconfiguration step (Observation 2) until \(P_1 = P_2\).

For each \(\ell_1, \ell_2 \in [r]\), if the relative order of \(\ell_1\) and \(\ell_2\) is the same in both \(P_1\) and \(P_2\), then note that Algorithm 2 does not change their relative order. Otherwise, Algorithm 2 performs an inversion: it swaps (or exchanges) them, inverting their relative order in \(P_1\). As every reconfiguration step corrects at most one such inversion (Observation 2), the number of reconfiguration steps required to reconfigure \(P_1\) to \(P_2\) is at least the number of inversions. The total number of inversions between \(P_1\) and \(P_2\) is in fact called Kendall’s Tau distance (or bubble sort distance), a well-known measure of dissimilarity between permutations [23]. This proves the optimality of Algorithm 2. \[\square\]

5.5 Circular-Arc Graphs

Definition 11 A circular-arc graph is the intersection graph of a set of arcs on a circle (Fig. 12).

Given a graph, its circular-arc representation can be constructed in linear time, if it exists [61, 62].

Theorem 6 SPR can be solved in polynomial time for circular-arc graphs.

**Proof** Let \((G, s, t, P, Q)\) be an SPR instance, where \(G\) is a circular-arc graph. If \(d(s, t) \leq 5\), then we can solve SPR in polynomial time (Theorem 7). If \(d(s, t) \geq 6\), then the \(s\)–\(t\) shortest path has at least 7 vertices. Note that the arcs corresponding to the middle vertices (at distance \([d(s, t)/2]\) from \(s\)) of \(P_1\) and \(P_2\) (say \(v\) and \(u\)) do not intersect any arc that intersects \(s\) or \(t\). Also, if we remove \(s\) and \(t\) and all their neighbours from the circle, \(^2\) the union of the remaining arcs (denoted by \(G \setminus (\text{nbhd}(s) \cup \text{nbhd}(t))\)) divides the circle into two disjoint arcs, which we can think of as two intervals. Now, we have two cases.

\(^2\) That is, remove the arcs corresponding to \(s\) and \(t\), and also all those arcs that intersect \(s\) or \(t\).
Fig. 12 A circular-arc graph
with \( d(s, t) = 4 \)

- Case 1: If the arcs \( v \) and \( u \) lie on the same arc of \( G \setminus (\text{nbhd}(s) \cup \text{nbhd}(t)) \), then we can think of the arcs as intervals of an interval graph. And since interval graphs are chordal, we know from [19] that SPR is polynomial-time solvable for them.
- Case 2: If the arcs \( v \) and \( u \) lie on different arcs of \( G \setminus (\text{nbhd}(s) \cup \text{nbhd}(t)) \), then \( P_1 \) and \( P_2 \) cannot be reconfigured. In particular, \( v \) can never be reconfigured to \( u \) because the two neighbours of \( v \) on \( P_1 \) are from one arc of \( G \setminus (\text{nbhd}(s) \cup \text{nbhd}(t)) \), and the two neighbours of \( u \) on \( P_2 \) are from the other arc of \( G \setminus (\text{nbhd}(s) \cup \text{nbhd}(t)) \).

This completes the proof.

\[ \blacksquare \]

5.6 Graphs of Bounded Diameter

Before we commence our theorem for this section, let us define the SPR graph of a given graph, an auxiliary graph created from an instance of SPR.

**Definition 12** (SPR Graph) Let \( G \) be a graph with two special vertices \( s \) and \( t \). Then the SPR graph of \( G \), denoted by \( \text{SPR}(G, s, t) \), is defined as follows.

\[
\begin{align*}
V(\text{SPR}(G, s, t)) &= \{ P \mid P \text{ is an } s-t \text{ shortest path in } G \}; \\
E(\text{SPR}(G, s, t)) &= \{(P, Q) \mid P \text{ can be reconfigured to } Q \text{ in one reconfiguration step}\}.
\end{align*}
\]

\( k \)-SPR\( (G, s, t) \) is similarly defined.

**Theorem 7** Let \( G \) be an \( n \)-vertex graph such that \( d(s, t) = c \). Then SPR can be solved in \( n^{O(c)} \) time for \( G \).

**Proof** The BFS tree from \( s \) to \( t \) has \( c \) layers, each layer with at most \( n \) vertices. Thus, the number of \( s-t \) shortest paths is at most \( n^c \). This means SPR\( (G, s, t) \) has at most \( n^c \) vertices and therefore at most \( n^{2c} \) edges. Note that two paths are reconfigurable in \( G \) if and only if their corresponding vertices are connected in SPR\( (G, s, t) \). And

\[ \blacksquare \]
therefore, given $G$, we can construct $\text{SPR}(G, s, t)$ and also check for the connectivity of two vertices in $\text{SPR}(G, s, t)$ in $n^{O(c)}$ time.

**Definition 13**

1. A **bipartite** graph is a graph whose vertex set can be partitioned into two independent sets.
2. A **split** graph is a graph whose vertex set can be partitioned into two sets: an independent set and a clique.
3. A **co-bipartite** graph is a graph whose vertex set can be partitioned into two cliques.

It is noteworthy that $\text{SPR}$ behaves differently on these three closely related graph classes. By Observation 1, $\text{SPR}$ is PSPACE-complete for bipartite graphs.

**Proof of Observation 1** Let $(G, s, t, P, Q)$ be an $\text{SPR}$ instance. Consider the layered BFS tree from $s$ to $t$, and delete all intra-layer edges (edges connecting two vertices in the same layer) from it, since these edges cannot be on an $s$–$t$ shortest path. The resulting graph $G'$ is bipartite: odd layer vertices form one partition, and even layer vertices form the other. $P$ and $Q$ are reconfigurable in $G'$ if and only if they are reconfigurable in $G$. This completes the proof. 

In contrast, $\text{SPR}$ is solvable in polynomial time for split graphs and co-bipartite graphs. The following fact about split graphs and co-bipartite graphs is well-known and easy to see [63].

**Observation 3** ([63]) The graph diameter of split graphs and co-bipartite graphs is at most 3.

This implies $d(s, t) \leq 3$ for split graphs and co-bipartite graphs, leading to the following corollary of Theorem 7.

**Corollary 2** $\text{SPR}$ can be solved in polynomial time for split graphs and co-bipartite graphs.

### 6 Gradation of the Complexity of $k$-SPR

In this section, we will see that for a fixed $n$, the complexity of $k$-SPR can decrease as $k$ increases, varying from 1 to $n$. Note that changing $k$ contiguous vertices requires a cycle of size $2k + 2$. We refer to such a cycle (which changes $k$ contiguous vertices) as a $k$-switch. We start with a simple observation (see Definition 12 to recall the definition of $k_1$-SPR$(G, s, t)$).

**Lemma 5** (Monotonicity of Diameter) Let $k_1$ and $k_2$ be two positive integers such that $k_1 \leq k_2$. Then the graph diameter of $k_1$-SPR$(G, s, t)$ is at least the graph diameter of $k_2$-SPR$(G, s, t)$.

**Proof** The vertex sets of the graphs $k_1$-SPR$(G, s, t)$ and $k_2$-SPR$(G, s, t)$ are the same (one vertex for each $s$–$t$ shortest path in $G$). Since every edge of $k_1$-SPR$(G, s, t)$ is also present in $k_2$-SPR$(G, s, t)$, this completes the proof. 

" Springer
Theorem 8 For every integer \( k \geq 1 \), there exists an \( n \)-vertex graph \( G \) such that the diameter of \( k \)-SPR\((G, s, t)\) is \( 2^{\Omega(n/k)} \).

Proof [64] showed that there exists an \( n \)-vertex graph \( G \) for which the diameter of SPR\((G, s, t)\) is \( 2^{\Omega(n)} \). Our graph is a simple modification of theirs.

Consider any odd \(^3 k \) (thus \( k = 2\ell + 1 \) for some \( \ell \)). Subdivide each edge \( \ell \) times (equivalently, replace each edge by a path \( P_{\ell+2} \) between its endpoints). Call this new graph \( G_\ell \). We make the following claim. \( \square \)

Claim 2 For all \( 1 \leq k' \leq k - 1 \), there is no \( k' \)-switch to reconfigure two \( s-t \) shortest paths in \( G_\ell \).

Note that every 1-switch in \( G \) was directly converted to a \( k \)-switch in \( G_\ell \). Further, the start and end vertices of a \( k' \)-switch (say \( u \) and \( v \)) have at least two neighbours in the next or previous layer, while all the newly added vertices in \( G_\ell \) have only one neighbour in the next layer and only one neighbour in the previous layer. Thus, \( u \) and \( v \) cannot be newly added vertices in \( G_\ell \), implying that they are vertices of the original graph \( G \).

Finally, we will show that \( d_G(u, v) = 2 \). If \( d_G(u, v) \geq 3 \), then \( d_{G_\ell}(u, v) \geq 3\ell + 3 \) (as every two vertices of \( G \) have distance at least \( \ell + 1 \) in \( G_\ell \)). And if \( d_G(u, v) \leq 1 \), then \( u \) and \( v \) are adjacent in \( G \), and the existence of two paths between them in \( G_\ell \) implies that there is a multiple edge \((u, v)\) in \( G \), a contradiction since our graphs are simple. This completes the proof of Claim 2.

Claim 2 implies that every switch in \( G_\ell \) is a \( k \)-switch which can be mapped back to a 1-switch in \( G \). Consider the two paths \( P_1 \) and \( P_2 \) in \( G \) which require \( 2^{\Omega(n)} \) 1-switches. These correspond to two paths \( P'_1 \) and \( P'_2 \) which require \( 2^{\Omega(n)} \) \( k \)-switches in \( G_\ell \).

It is easy to see that the degree of each vertex in the graph \( G \) from [64] is upper-bounded by a constant. Thus, \( G \) has \( O(n) \) edges. Since each edge of \( G \) is subdivided \( \ell \) times in \( G_\ell \), the number of vertices in \( G_\ell \) is \( N = O(n\ell) = O(nk) \). Hence, the diameter of \( k \)-SPR\((G_\ell, s, t)\) is at least \( 2^{\Omega(n)} = 2^{\Omega(N/k)} \).

Figure 13 illustrates known complexity results for \( k \)-SPR as \( k \) varies from 1 to \( n \), for a fixed value of \( n \). Lemma 1 implies that \( k \)-SPR is \( \mathsf{PSPACE} \)-complete when \( k = O(1) \).

Theorem 9 \( k \)-SPR can be solved in polynomial time when \( k \geq n/2 \).

Proof We can split the proof into two cases.

Case 1: \( d(s, t) < n/2 \). This means \( d(s, t) < n/2 \leq k \). Since we are allowed to reconfigure up to \( k \) contiguous vertices in one reconfiguration step of \( k \)-SPR, we can trivially reconfigure \( P_1 \) to \( P_2 \) in polynomial time.

Case 2: \( d(s, t) \geq n/2 \). This means at least \( n/2 \) vertices of the graph lie on \( P_1 \). So \( P_2 \) contains at most \( n/2 \) vertices that do not lie on \( P_1 \). Since \( k \geq n/2 \), we can trivially reconfigure \( P_1 \) to \( P_2 \) in polynomial time. \( \square \)

\(^3\) We assume \( k \) is odd for simplicity. It is easy to see that our proof works for even values of \( k \) too.
7 Optimization Variants of SPR

In this section, we define three variants of the Shortest Path Reconfiguration problem. In this setting, we are allowed to change any number of vertices at a time. In addition, we pay a price of \( p_i \) for changing \( i \) vertices on a path. Furthermore,

\[ p_1 \leq p_2 \leq \cdots \leq p_{n-1} \leq p_n. \]

**Definition 14** (MinSumSPR) Given \((G, s, t, P_1, P_2)\), an SPR instance, output a reconfiguration sequence from \( P_1 \) to \( P_2 \) (if it exists) that minimises the total cost of reconfiguration.

**Definition 15** (MinMaxSPR) Given \((G, s, t, P_1, P_2)\), an SPR instance, output a reconfiguration sequence from \( P_1 \) to \( P_2 \) (if it exists) that minimises the maximum cost of reconfiguration.

Generalizing MinSumSPR and MinMaxSPR, we get the following.

**Definition 16** (MinTop-\( \ell \)-SPR) Given \((G, s, t, P_1, P_2)\), an SPR instance, output a reconfiguration sequence from \( P_1 \) to \( P_2 \) (if it exists) that minimises the sum total of the maximum \( \ell \) (or top-\( \ell \)) costs of reconfiguration.

Note that MinSumSPR is a special case of MinTop-\( \ell \)-SPR with \( \ell = \infty \) and MinMaxSPR is a special case of MinTop-\( \ell \)-SPR with \( \ell = 1 \).

**Lemma 6** For every \( n \)-vertex graph \( G \), the diameter of \( k \)-SPR\((G, s, t)\) is at most \( 2^n \).

**Proof** Each \( s \rightarrow t \) shortest path in \( k \)-SPR\((G, s, t)\) is a distinct subset of vertices of \( G \). As \( G \) has \( n \) vertices, the \( k \)-SPR graph has at most \( 2^n \) vertices. \( \square \)

**Theorem 10** There is no polynomial-time algorithm that approximates MinTop-\( \ell \)-SPR to within a factor of \( O(2^{n^2}) \), unless \( \text{PSPACE} = \text{P} \).
Proof For the sake of contradiction, assume that there is an $O(2n^2)$-factor approximation algorithm for MinTop-$\ell$-SPR. We reduce the original SPR problem, which is known to be PSPACE-complete [19], to MinTop-$\ell$-SPR. Let $I = (G = (V, E), s, t, P_1, P_2)$ be an instance of SPR.

We assign the following costs for changing $i$ contiguous vertices, denoted by $p_i$.

$$p_i = \begin{cases} 1, & i = 1; \\ \ell \cdot 2^{n^2+1}, & i \geq 2. \end{cases}$$

If $I \in$ SPR, there exists a reconfiguration sequence which changes one vertex per reconfiguration step. The number of such steps is at most diam(SPR($G$, $s$, $t$)), thus OPT pays at most $\ell$ since all moves are changing 1 vertex, and the cost is 1.

Thus, the cost paid by OPT is at most $2n$. Since ALG is an $O(2n^2)$-approximation algorithm, $\text{ALG} \leq 2n$.

Conversely, assume that $I \not\in$ SPR. Then every reconfiguration sequence must change at least 2 vertices in at least 1 reconfiguration step. Thus, in the MinTop-$\ell$-SPR instance, we pay a cost of $\ell \cdot 2^{n^2+1}$ at least once. As OPT is lower-bounded by this quantity, $\text{ALG} \geq \ell \cdot 2^{n^2+1}$.

Algorithm 3 (REDUC) explains the reduction formally. The correctness of Algorithm 3 is shown by the above statements. The reduction is polynomial time, because writing $2^{n^2+1} \cdot 2n^2$ takes at most poly($n$) bits, and ALG runs in polynomial time. Thus, if there is an $O(2n^2)$-approximation for MinTop-$\ell$-SPR, then we can solve SPR in polynomial time, implying that PSPACE = P.

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input:} An instance of SPR
\State 1: Creates MinTop-$\ell$ SPR, by setting costs as described
\State 2: Run ALG on this instance
\State 3: \textbf{if } $c(\text{ALG}) \leq 2n^2 + 1 \cdot \ell$ \textbf{then}
\State 4: \hspace{1em} Output $I \in$ SPR
\State 5: \textbf{else}
\State 6: \hspace{1em} Output $I \not\in$ SPR
\State 7: \textbf{end if}
\end{algorithmic}
\end{algorithm}

Corollary 3 There is no polynomial-time algorithm that approximates MinSumSPR to within a factor of $O(2n^2)$, unless PSPACE = P.

Corollary 4 There is no polynomial-time algorithm that approximates MinMaxSPR to within a factor of $O(2n^2)$, unless PSPACE = P.
8 Length versus Number of Shortest Paths

In this section, we study how the number of vertices in $\text{SPR}(G,s,t)$ varies with the path length between $s$ and $t$. We denote $|V_{\text{SPR}}|$ as a function $f$ that takes $d(s,t)$ as input.

It is easy to see that $f(x) = O(2^n)$ (Lemma 6) for all values of $d(s,t)$. For specific values of $d(s,t)$, we have the following stronger bounds, represented by Fig. 14.

**Lemma 7**

\[
d(s,t) = x = n/2 \quad \Rightarrow \quad f(x) = \Theta(2^n);
\]
\[
d(s,t) = x = \Theta(\sqrt{n}) \quad \Rightarrow \quad f(x) = \Omega\left(2^{\sqrt{n}\log\sqrt{n}}\right);
\]
\[
d(s,t) = x = n - O(1) \quad \Rightarrow \quad f(x) = O(1).
\]

**Proof** Consider the following gadget graph on $l + 2$ vertices. Two independent vertices $u$ (called the start point of the gadget) and $v$ (called the endpoint of the gadget) are both connected to a set of $l$ independent vertices. Identifying the endpoint of the gadget with the start point of a second gadget adds $l + 1$ more vertices. If we chain $g$ gadgets in series in this way, the resulting graph has $1 + g(l + 1)$ vertices.

Let $s$ and $t$ be the start point of the first gadget and the end point of the last gadget, respectively. The graph has $n = 1 + g(l + 1)$ vertices, $d(s,t) = 2g = \Theta(n)$, and the number of $s$-$t$ shortest paths is $l^8$. When $l = c$ for some constant $c$, we get $c^{\Theta(l)} = \Theta(2^n)$. Similarly, for $l = \Theta(\sqrt{n})$, $g = \Theta(\sqrt{n})$, we get path length of $\Theta(\sqrt{n})$ and roughly $\sqrt{n^{\sqrt{n}}} = \Theta(2^{\sqrt{n}\log n})$ paths. This shows the first two implications.

Let $d(s,t) = n - c$, for some constant $c$. These $c$ vertices are present in at most $c$ distinct layers, and each layer has at most $c$ vertices. Thus, the number of possible reconfigurations is at most $c^2$, which is a constant. This shows the third implication. \qed
9 Conclusion

We conclude with some possible directions for future research on SPR and $k$-SPR.

- **Experimental results:** This entire paper contains only theoretical results. One can try to simulate or observe the behaviour of SPR (or $k$-SPR) on practical instances, and see if they yield something more substantial or meaningful than what we could obtain theoretically.

- **A graph invariant of SPR:** Since SPR is $\text{PSPACE}$-hard in general, we study SPR on specific graph classes and provide polynomial-time algorithms for some of them, each one requiring its own separate proof. Is there a graph invariant (like treewidth) that characterizes the complexity of SPR?

- **SPR for unit disk graphs:** A highly practical application of SPR is in rerouting messages across a network of telecommunication towers. Each tower has a coverage radius (known as a range), and two towers can interact if one lies in the range of the other. When one tower fails (this could happen, for instance, due to a high rate of incoming messages at the tower, leading to an overload), we need to quickly reroute messages using a different set of towers, which is precisely the SPR problem for unit disk graphs. This is seemingly a very difficult problem, and even an approximation algorithm for it will be very useful.

- **Parameterized complexity:** Although both SPR and $k$-SPR are $\text{PSPACE}$-complete, they might be fixed-parameter tractable, for the right choice of the parameter. It will be interesting to explore which parameters could work (if any).

- **A novel variant of reconfigurability:** What if shortest paths need not change in $k$ contiguous vertices at a time? A variant of reconfigurability could be defined as follows: One shortest path can “hop” onto another shortest path, whenever it finds that the $(i+1)$-th vertex of the other path is adjacent to its $i$-th vertex. This notion of reconfigurability is also practical, as it allows shortest paths to directly use other shortest paths that already exist.

**Acknowledgements**  Agastya Vibhuti Jha would like to thank Dr. Jatin Batra for introducing him to Ordered Optimization, which led to the idea of $k$-SPR.

**Funding**  Open access publishing supported by the National Technical Library in Prague.

**Declarations**

**Conflict of interest**  The authors are unaware of any potential Conflict of interest to disclose except their current and previous affiliations. Kshitij Gajjar did his PhD from the Tata Institute of Fundamental Research, India, and postdocs at the Technion - Israel Institute of Technology, Israel, and the National University of Singapore, Singapore. Abhiruk Lahiri did his PhD at the Indian Institute of Science, India, and postdocs at the University of Haifa, Israel, and Ariel University, Israel. No animals or humans were harmed while carrying out this research.

**Human and Animal Rights**  No animals or humans were harmed while carrying out this research.

**Open Access**  This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included.
in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Gajjar, K., Jha, A.V., Kumar, M., Lahiri, A.: Reconfiguring shortest paths in graphs. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Virtual Event, February 22–March 1, 2022, pp. 9758–9766. AAAI Press, USA. https://ojs.aaai.org/index.php/AAAI/article/view/21211 (2022)

2. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the N × N extension of the 15-puzzle is intractable. In: Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science, USA, pp. 168–172. http://www.aaai.org/Library/AAAI/1986/aaai86-027.php (1986)

3. Goldreich, O.: Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard. In: Studies in Complexity and Cryptography. Miscellanea on the Interplay Between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman. Lecture Notes in Computer Science, vol. 6650, pp. 1–5. Springer, USA. https://doi.org/10.1007/978-3-642-22670-0_1 (2011)

4. Demaine, E.D., Demaine, M.L., Eisenstat, S., Lubiw, A., Winslow, A.: Algorithms for solving rubik's cubes. In: Algorithms–ESA 2011–19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6942, pp. 689–700. https://doi.org/10.1007/978-3-642-23719-5_58 (2011)

5. Demaine, E.D., Eisenstat, S., Rudoy, M.: Solving the rubik's cube optimally is np-complete. In: 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France. LIPIcs, vol. 96, pp. 24–12413. Dagstuhl, Germany. https://doi.org/10.4230/LIPIcs.STACS.2018.24 (2018)

6. Mouawad, A.E., Nishimura, N., Pathak, V., Raman, V.: Shortest reconfiguration paths in the solution space of Boolean formulas. SIAM J. Discret. Math. 31(3), 2185–2200 (2017)

7. Haas, R., Seyffarth, K.: Reconfiguring dominating sets in some well-covered and other classes of graphs. Discrete Math. 340(8), 1802–1817 (2017)

8. Lokshhtanov, D., Mouawad, A.E.: The complexity of independent set reconfiguration on bipartite graphs. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, February 28 to March 3, 2018, New Orleans, LA, USA, January 7-10, 2018, pp. 185–195. SIAM, USA. https://doi.org/10.1137/1.9781611975031.13 (2018)

9. Wrochna, M.: Reconfiguration in bounded bandwidth and tree-depth. J. Comput. Syst. Sci. 93, 1–10 (2018)

10. Lokshhtanov, D., Mouawad, A.E., Panolan, F., Ramanujan, M.S., Saurabh, S.: Reconfiguration on sparse graphs. J. Comput. Syst. Sci. 95, 122–131 (2018)

11. Nishimura, N.: Introduction to reconfiguration. Algorithms 11(4), 52 (2018)

12. Mouawad, A.: On reconfiguration problems: Structure and tractability. PhD thesis, University of Waterloo. http://hdl.handle.net/10012/9183 (2015)

13. Mouawad, A.E., Nishimura, N., Ramai, V., Simjour, N., Suzuki, A.: On the parameterized complexity of reconfiguration problems. Algorithmica 78(1), 274–297 (2017)

14. Mouawad, A.E., Nishimura, N., Raman, V., Siebertz, S.: Vertex cover reconfiguration and beyond. Algorithms 11(2), 20 (2018)

15. Kaminski, M., Medvedev, P., Milanic, M.: Shortest paths between shortest paths and independent sets. In: Combinatorial Algorithms - 21st International Workshop, IWOCA 2010, July 26-28, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6460, pp. 56–67. London (2010)

16. Hopcroft, J.E., Schwartz, J.T., Sharir, M.: On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “Warehousman’s Problem”. Int. J. Robot. Res. 3(4), 76–88 (1984)

17. Hopcroft, J.E., Wilfong, G.T.: Reducing multiple object motion planning to graph searching. SIAM J. Comput. 15(3), 768–785 (1986)
18. Hopcroft, J.E., Joseph, D., Whitesides, S.: Movement problems for 2-dimensional linkages. SIAM J. Comput. 13(3), 610–629 (1984)
19. Bonsma, P.S.: The complexity of rerouting shortest paths. Theor. Comput. Sci. 510, 1–12 (2013)
20. Bonsma, P.S.: Rerouting shortest paths in planar graphs. Discret. Appl. Math. 231, 95–112 (2017)
21. Asplund, J., Edoh, K.D., Haas, R., Hristova, Y., Novick, B., Werner, B.: Reconfiguration graphs of shortest paths. Discret. Math. 341(10), 2938–2948 (2018)
22. Asplund, J., Werner, B.: Classification of reconfiguration graphs of shortest path graphs with no induced $4$-cycles. Discret. Math. 343(1), 111640 (2020)
23. Sedgewick, R., Wayne, K.: Algorithms. Addison-Wesley, New York (2016)
24. Siggers, M.H.: Graphs for which the homomorphism extension reconfiguration problem is trivial. arXiv:abs/2208.04071 (2022)
25. Gavril, F.: Algorithms for a maximum clique and a maximum independent set of a circle graph. Networks 3(3), 261–273 (1973)
26. Transport, R.: Review of Maritime Transport, United Nations Conference on Trade and Development. Accessed 8th Sep 2021. https://unctad.org/webflyer/review-maritime-transport-2018 (2018)
27. Co-operation, O., Development: Organisation for Economic Co-operation and Development, Ocean shipping and shipbuilding. https://www.oecd.org/ocean/topics/ocean-shipping/ (2021). Accessed 8 Sept 2021
28. Wilson, I.D., Roach, P.A.: Container stowage planning: a methodology for generating computerised solutions. J. Oper. Res. Soc.: 51(11), 1248–1255 (2000)
29. Avriel, M., Penn, M., Shiprer, N., Witteboon, S.: Stowage planning for container ships to reduce the number of shifts. Ann. Oper. Res. 76, 55–71 (1998)
30. Avriel, M., Penn, M., Shiprer, N.: Container ship stowage problem: complexity and connection to the coloring of circle graphs. Discret. Appl. Math. 103(1–3), 271–279 (2000)
31. Tierney, K., Pacino, D., Jensen, R.M.: On the complexity of container stowage planning problems. Discret. Appl. Math. 169, 225–230 (2014)
32. Gajjar, K., Radhakrishnan, J.: Distance-preserving subgraphs of interval graphs. In: 25th Annual European Symposium on Algorithms (ESA 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 87, pp. 39–13913. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany. http://drops.dagstuhl.de/opus/volltexte/2017/7879 (2017)
33. Caserta, M., Voß, S., Sniedovich, M.: Applying the corridor method to a blocks relocation problem. OR Spectr. 33(4), 915–929 (2011)
34. Caserta, M., Schwarze, S., Voß, S.: A mathematical formulation and complexity considerations for the blocks relocation problem. Eur. J. Oper. Res. 219(1), 96–104 (2012)
35. Dahlhaus, E., Horák, P., Miller, M., Ryan, J.F.: The train marshalling problem. Discret. Appl. Math. 103(1–3), 41–54 (2000)
36. Jaehn, F., Rieder, J., Wiehl, A.: Minimizing delays in a shunting yard. OR Spectr. 37(2), 407–429 (2015)
37. Rinaldi, F., Rimoli, R.: Solving the train marshalling problem by inclusion-exclusion. Discret. Appl. Math. 217, 685–690 (2017)
38. Dörrpinghaus, J., Schrader, R.: A graph-theoretic approach to the train marshalling problem. In: Proceedings of the 2018 Federated Conference on Computer Science and Information Systems, FedCSIS 2018, Poznań, Poland, September 9-12, 2018. Annals of Computer Science and Information Systems, vol. 15, pp. 227–231. Poland. https://doi.org/10.15439/2018F26 (2018)
39. Falsafain, H., Tamannaei, M.: A novel dynamic programming approach to the train marshalling problem. IEEE Trans. Intell. Transp. Syst. 21(2), 701–710 (2020)
40. Even, S., Itai, A.: Queues stacks and graphs. In: Theory of Machines and Computations: Proceedings of an International Symposium on the Theory of Machines and Computations, pp. 71–86. Academic Press, USA. https://doi.org/10.1016/B978-0-12-417750-5.50011-7 (1971)
41. Even, S., Pnueli, A., Lempel, A.: Permutation graphs and transitive graphs. J. ACM 19(3), 400–410 (1972)
42. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, USA. https://doi.org/10.1016/C2013-0-10739-8 (1980)
43. Brandstädt, A., Le, V.B., Spinrad, J.P.: Discrete Mathematics and Applications. SIAM, USA. https://doi.org/10.1137/1.9780898719796 (1999)
44. Valiant, L.G., Brebner, G.J.: Universal schemes for parallel communication. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA, pp. 263–277. ACM, USA. https://doi.org/10.1145/800076.802479 (1981)
45. Hayes, J.P., Mudge, T.N., Stout, Q.F., Colley, S., Palmer, J.: A microprocessor-based hypercube supercomputer. IEEE Micro 6(5), 6–17 (1986)
46. Stamoulis, G.D., Tsitsiklis, J.N.: The efficiency of greedy routing in hypercubes and butterflies. IEEE Trans. Commun. 42(11), 3051–3061 (1994)
47. Pifarré, G.D., Gravano, L., Felépin, S.A., Sanz, J.L.C.: Fully adaptive minimal deadlock-free packet routing in hypercubes, meshes, and other networks: algorithms and simulations. IEEE Trans. Parallel Distrib. Syst. 5(3), 247–263 (1994)
48. Grammatikakis, M.D., Hsu, D.F., Sibeyn, J.F.: Packet routing in fixed-connection networks: a survey. J. Parallel Distrib. Comput. 54(2), 77–132 (1998)
49. Greenberg, A.G., Hajek, B.E.: Deflection routing in hypercube networks. IEEE Trans. Commun. 40(6), 1070–1081 (1992)
50. Wang, S., Djahel, S., Zhang, Z., McManis, J.: Next road rerouting: a multiagent system for mitigating unexpected urban traffic congestion. IEEE Trans. Intell. Transp. Syst. 17(10), 2888–2899 (2016)
51. Bast, H., Funke, S., Matijevic, D.: Ultrafast shortest-path queries via transit nodes. In: The Shortest Path Problem. Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November 13-14, 2006. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 74, pp. 175–192. https://doi.org/10.1090/dimacs/074/07 (2006)
52. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant time shortest-path queries in road networks. In: Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007. SIAM. https://doi.org/10.1137/1.9781611972870.5 (2007)
53. Bauer, R., Delling, D.: SHARC: fast and robust unidirectional routing. ACM J. Exp. Algorithm 14 (2009)
54. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for a*: Efficient point-to-point shortest path algorithms. In: Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments, ALENEX 2006, Miami, Florida, USA, January 21, 2006, pp. 129–143. SIAM. https://doi.org/10.1137/1.9781611972863.13 (2006)
55. Kirkpatrick, D., Liu, P.: Characterizing minimum-length coordinated motions for two discs. In: Proceedings of the 28th Canadian Conference on Computational Geometry, CCCG 2016, August 3-5, 2016, Simon Fraser University, Vancouver, British Columbia, Canada, Canada, pp. 252–259 (2016)
56. Demaine, E.D., Fekete, S.P., Keldenich, P., Meijer, H., Scheffer, C.: Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. SIAM J. Comput. 48(6), 1727–1762 (2019)
57. Soltan, V., Chepoi, V.: Conditions for invariance of set diameters under d-convexification in a graph. Cybernetics 19(6), 750–756 (1983)
58. Farber, M., Jamison, R.E.: On local convexity in graphs. Discret. Math. 66(3), 231–247 (1987)
59. Bandelt, H., Chepoi, V.: A helly theorem in weakly modular space. Discret. Math. 160(1–3), 25–39 (1996)
60. Chepoi, V.: Classification of graphs by means of metric triangles. Methdy Diskret. Analiz. 96, 75–93 (1989)
61. Tucker, A.: An efficient test for circular-arc graphs. SIAM J. Comput. 9(1), 1–24 (1980)
62. Kaplan, H., Nussbaum, Y.: A simpler linear-time recognition of circular-arc graphs. Algorithmica 61(3), 694–737 (2011)
63. Graphclass: Information System on Graph Classes and their Inclusions. https://www.graphclasses.org/classes/gc_1242.html (2021). Accessed 11 Sept 2021
64. Kamiński, M., Medvedev, P., Milanic, M.: Shortest paths between shortest paths. Theor. Comput. Sci. 412(39), 5205–5210 (2011)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.