Shortest Two Disjoint Paths in Conservative Graphs

Ildikó Schlotter
Centre for Economic and Regional Studies, Budapest, Hungary
Budapest University of Technology and Economics, Hungary

Abstract
We consider the following problem that we call the Shortest Two Disjoint Paths problem: given an undirected graph $G = (V,E)$ with edge weights $w : E \to \mathbb{R}$, two terminals $s$ and $t$ in $G$, find two internally vertex-disjoint paths between $s$ and $t$ with minimum total weight. As shown recently by Schlotter and Sebő (2022), this problem becomes NP-hard if edges can have negative weights, even if the weight function is conservative, i.e., there are no cycles in $G$ with negative total weight. We propose a polynomial-time algorithm that solves the Shortest Two Disjoint Paths problem for conservative weights in the case when the negative-weight edges form a constant number of trees in $G$.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Shortest paths; Theory of computation → Dynamic programming

Keywords and phrases Shortest paths, disjoint paths, conservative weights, graph algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.57

Funding Ildikó Schlotter: supported by the Hungarian Academy of Sciences under its Momentum Programme (LP2021-2) and its János Bolyai Research Scholarship.

1 Introduction
Finding disjoint paths between given terminals is a fundamental problem in algorithmic graph theory and combinatorial optimization. Besides its theoretical importance, it is also motivated by numerous applications in transportation, VLSI design, and network routing. In the Disjoint Paths problem, we are given $k$ terminal pairs $(s_i, t_i)$ for $i \in \{1, \ldots, k\}$ in an undirected graph $G$, and the task is to find pairwise vertex-disjoint paths $P_1, \ldots, P_k$ so that $P_i$ connects $s_i$ with $t_i$ for each $i \in \{1, \ldots, k\}$. This problem was shown to be NP-hard by Karp [7] when $k$ is part of the input, and remains NP-hard even on planar graphs [10]. Robertson and Seymour [11] proved that there exists an $f(k)n^3$ algorithm for Disjoint Paths with $k$ terminal pairs, where $n$ is the number of vertices in $G$ and $f$ some computable function; this celebrated result is among the most important achievements of graph minor theory. In the Shortest Disjoint Paths problem we additionally require that $P_1, \ldots, P_k$ have minimum total length (in terms of the number of edges). For fixed $k$, the complexity of this problem is one of the most important open questions in the area. Even the case for $k = 2$ had been open for a long time, until Björklund and Husfeldt [3] gave a randomized polynomial-time algorithm for it in 2019. For directed graphs the problem becomes much harder: the Directed Disjoint Paths problem is NP-hard already for $k = 2$. The Disjoint Paths problem and its variants have also received considerable attention when restricted to planar graphs [6, 5, 8, 1, 4, 14, 9].

The variant of Disjoint Paths when $s_1 = \cdots = s_k = s$ and $t_1 = \cdots = t_k = t$ is considerably easier, since one can find $k$ pairwise (openly vertex- or edge-) disjoint paths between $s$ and $t$ using a max-flow computation. Applying standard techniques for computing...
a minimum-cost flow (see e.g. [15]), one can even find $k$ pairwise disjoint paths between $s$ and $t$ with minimum total weight, given non-negative weights on the edges. Notice that if negative weights are allowed, then flow techniques break down for undirected graphs: in order to construct an appropriate flow network based on our undirected graph $G$, the standard technique is to direct each edge of $G$ in both directions; however, if edges can have negative weight, then this operation creates negative cycles consisting of two arcs, an obstacle for computing a minimum-cost flow. Recently, Schlotter and Sebő [13] have shown that this issue is a manifestation of a complexity barrier: finding two openly disjoint paths with minimum total weight between two vertices in an undirected edge-weighted graph is \textit{NP}-hard, even if weights are \textit{conservative} (i.e., no cycle has negative total weight) and each edge has weight in $\{-1, 1\}$.

Note that negative edge weights occur in network problems due to various reasons: for example, they might arise as a result of some reduction (e.g., deciding the feasibility of certain scheduling problems with deadlines translates into finding negative-weight cycles), or as a result of data that is represented on a logarithmic scale. We remark that the \textsc{Single-Source Shortest Paths} problem is the subject of active research for the case when negative edges are allowed; see Bernstein et al. [2] for an overview of the area and their state-of-the-art algorithm running in near-linear time on directed graphs.

\textbf{Our contribution}

We consider the following problem which concerns finding paths between two fixed terminals (as opposed to the classic \textsc{Shortest Disjoint Paths} problem):

\begin{verbatim}
SHORTEST TWO DISJOINT PATHS:
Input: An undirected graph $G = (V,E)$, a weight function $w: E \rightarrow \mathbb{R}$ that is conservative on $G$, and two vertices $s$ and $t$ in $G$.
Task: Find two paths $P_1$ and $P_2$ between $s$ and $t$ with $V(P_1) \cap V(P_2) = \{s,t\}$ that minimizes $w(P_1) + w(P_2)$.
\end{verbatim}

A \textit{solution} for an instance $(G, w, s, t)$ of \textsc{Shortest Two Disjoint Paths} is a pair of $(s, t)$-paths that are openly disjoint, i.e., do not share vertices other than their endpoints.

From the \textit{NP}-hardness proof for \textsc{Shortest Two Disjoint Paths} by Schlotter and Sebő [13] it follows that the problem remains \textit{NP}-hard even if the set of negative-weight edges forms a perfect matching. Motivated by this intractability, we focus on the “opposite” case when the subgraph of $G$ spanned by the set $E^- = \{e \in E : w(e) < 0\}$ of negative-weight edges, denoted by $G[E^-]$, has only few connected components.\(^2\) Note that since $w$ is conservative on $G$, the graph $G[E^-]$ is acyclic. Hence, if $c$ denotes the number of connected components in $G[E^-]$, then $G[E^-]$ in fact consists of $c$ trees.

We can think of our assumption that $c$ is constant as a compromise for allowing negative-weight edges but requiring that they be confined to a small part of the graph. For a motivation, consider a network where negative-weight edges arise as some rare anomaly. Such an anomaly may occur when, in a certain part of a computer network, some information can be collected while traversing the given edge. If such information concerns, e.g., the detection of (possibly) faulty nodes or edges in the network, then it is not unreasonable to assume that these faults are concentrated to a certain part of the network, due to underlying physical causes that are responsible for the fault.

\(^1\) In fact, Schlotter and Sebő use an equivalent formulation of the problem where, instead of finding two openly disjoint paths between $s$ and $t$, the task is to find two vertex-disjoint paths between $\{s_1, s_2\}$ and $\{t_1, t_2\}$ for four vertices $s_1, s_2, t_1, t_2 \in V$.

\(^2\) See Section 2 for the precise definition of a subgraph spanned by an edge set.
Ideally, one would aim for an algorithm that is fixed-parameter tractable (FPT) when parameterized by $c$; however, already the case $c = 1$ turns out to be challenging. We prove the following result, which can be thought of as a first step towards an FPT algorithm:

**Theorem 1.** For each constant $c \in \mathbb{N}$, Shortest Two Disjoint Paths can be solved in polynomial time on instances where the set of negative edges spans $c$ trees in $G$.

Our algorithm first applies standard flow techniques to find minimum-weight solutions among those that have a simple structure in the sense that there is no negative tree in $G[E^-]$ used by both paths. To deal with more complex solutions where there is at least one tree $T$ in $G[E^-]$ used by both paths, we use recursion to find two openly disjoint paths from $s$ to $T$, and from $T$ to $t$; to deal with the subpaths of the solution that heavily use negative edges from $T$, we apply an intricate dynamic programming method that is based on significant insight into the structural properties of such solutions.

**Organization**

We give all necessary definitions in Section 2. In Section 3 we make initial observations about optimal solutions for an instance $(G, w, s, t)$ of Shortest Two Disjoint Paths, and we also present a lemma of key importance that will enable us to create solutions by combining partial solutions that are easier to find (Lemma 10). We present the algorithm proving our main result, Theorem 1, in Section 4. In Section 4.1 we give a general description of our algorithm, and explain which types of solutions can be found using flow-based techniques. We proceed in Section 4.2 by establishing structural observation that we need to exploit in order to find those types of solutions where more advanced techniques are necessary. Section 4.3 contains our dynamic programming method for finding partial solutions which, together with Lemma 10, form the heart of our algorithm. We conclude with some questions for further research in Section 5. All proofs are deferred to the full version of our paper [12].

**2 Notation**

For a positive integer $\ell$, we use $[\ell] := \{1, 2, \ldots, \ell\}$.

Let a graph $G$ be a pair $(V, E)$ where $V$ and $E$ are the set of vertices and edges, respectively. For two vertices $u$ and $v$ in $V$, an edge connecting $u$ and $v$ is denoted by $uv$ or $vu$.

For a set of $X$ of vertices (or edges), let $G - X$ denote the subgraph of $G$ obtained by deleting the vertices (or edges, respectively) of $X$; if $X = \{x\}$ then we may simply write $G - x$ instead of $G - \{x\}$. Given a set $F \subseteq E$ of edges in $G$, we denote by $V(F)$ the vertices incident to some edge of $F$. The subgraph of $G$ spanned by $F$ is the graph $(V(F), F)$; we denote this subgraph as $G[F]$.

A walk $W$ in $G$ is a series $e_1, e_2, \ldots, e_\ell$ of edges in $G$ for which there exist vertices $v_0, v_1, \ldots, v_\ell$ in $G$ such that $e_i = v_{i-1}v_i$ for each $i \in [\ell]$; note that both vertices and edges may appear repeatedly on a walk. We denote by $V(W)$ the set of vertices contained by or appearing on $W$, that is, $V(W) = \{v_0, v_1, \ldots, v_\ell\}$. The endpoints of $W$ are $v_0$ and $v_\ell$, or in other words, it is a $(v_0, v_\ell)$-walk, while all vertices on $W$ that are not endpoints are inner vertices. If $v_0 = v_\ell$, then we say that $W$ is a closed walk.

A path is a walk on which no vertex appears more than once. By a slight abuse of notation, we will usually treat a path as a set $\{e_1, e_2, \ldots, e_\ell\}$ of edges for which there exist distinct vertices $v_0, v_1, \ldots, v_\ell$ in $G$ such that $e_i = v_{i-1}v_i$ for each $i \in [\ell]$. For any $i$ and $j$ with $0 \leq i < j \leq \ell$ we will write $P[v_i, v_j]$ for the subpath of $P$ between $v_i$ and $v_j$, consisting of edges $e_{i+1}, \ldots, e_j$. Note that since we associate no direction with $P$, we have
Shortest Two Disjoint Paths in Conservative Graphs

Given two vertices $s$ and $t$, an $(s, t)$-path is a path whose endpoints are $s$ and $t$. Similarly, for two subsets $S$ and $T$ of vertices, an $(S, T)$-path is a path with one endpoint in $S$ and the other endpoint in $T$.

We say that two paths are *vertex- or edge-disjoint*, if they do not share a common vertex or edge, respectively. Two paths are *openly disjoint*, if they share no common vertices apart from possibly their endpoints. Given vertices $s_1, s_2, t_1,$ and $t_2$, we say that two $\{(s_1, s_2), \{t_1, t_2\}\}$-paths are *permissively disjoint*, if a vertex $v$ can only appear on both paths if either $v = s_1 = s_2$ or $v = t_1 = t_2$. Two paths properly intersect, if they share at least one edge, but neither is the subpath of the other.

A cycle in $G$ is a set $\{e_1, e_2, \ldots, e_\ell\}$ of distinct edges in $G$ such that $e_1, e_2, \ldots, e_{\ell-1}$ form a path in $G - e_\ell$ whose endpoints are connected by $e_\ell$. A set $T \subseteq E$ of edges in $G$ is connected, if for every pair of edges $e$ and $e'$ in $T$, there is a path contained in $T$ containing both $e$ and $e'$. If $T$ is connected and acyclic, i.e., contains no cycle, then $T$ is a tree in $G$. Given two vertices $a$ and $b$ in a tree $T$, we denote by $T[a, b]$ the unique path contained in $T$ whose endpoints are $a$ and $b$. For an edge $uv \in T$ and a path $P$ within $T$ such that $uv \notin P$, we say that $v$ is closer to $P$ in $T$ than $u$, if $v \in V(T[u, p])$ for some vertex $p \in V(P)$.

Given a weight function $w : E \to \mathbb{R}$ on the edge set of $G$, we define the *weight* of any edge set $F \subseteq E$ as $w(F) = \sum_{e \in F} w(e)$. We extend this notion for any pair $F = (F_1, F_2)$ of edge sets by letting $w(F) = w(F_1) + w(F_2)$. The restriction of $w$ to an edge set $F \subseteq E$, i.e., the function whose domain is $F$ and has value $w(f)$ on each $f \in F$, is denoted by $w|_F$. We say that $w$ (or, to make the dependency on $G$ explicit, the weighted graph $(G, w)$) is conservative, if no cycle in $G$ has negative total weight.

### 3 Structural Observations

Let $G = (V, E)$ be an undirected graph with a conservative weight function $w : E \to \mathbb{R}$. Let $E^- = \{e \in E : w(e) < 0\}$ denote the set of negative edges, and $T$ the set of negative trees they form. More precisely, let $T$ be the set of connected components in the subgraph $G[E^-]$; the acyclicity of each $T \in T$ follows from the conservativeness of $w$. For any subset $T'$ of $T$, we use the notation $E(T') = \bigcup_{T \in T'} E(T)$ and $V(T') = \bigcup_{T \in T'} V(T)$.

In Section 3.1 we gather a few useful properties of conservative weight functions. In Section 3.2 we collect observations on how an optimal solution can use different trees in $T$. We close the section with a lemma of key importance in Section 3.3 that enables us to compose solutions by combining two path pairs without violating our requirement of disjointness.

### 3.1 Implications of Conservative Weights

The next two lemmas establish implications of the conservativeness of our weight function. Lemma 2 concerns closed walks, while Lemma 3 considers paths running between two vertices on some negative tree in $T$. These lemmas will be useful in proofs where a given hypothetical solution is “edited” – by removing certain subpaths from it and replacing them with paths within some negative tree – in order to obtain a specific form without increasing its weight.

**Lemma 2.** If $W$ is a closed walk that does not contain any edge with negative weight more than once, then $w(W) \geq 0$.

**Lemma 3.** Let $x, y, x', y'$ be four distinct vertices on a tree $T$ in $T$.

1. If $Q$ is an $(x, y)$-walk in $G$ using each edge of $E^-$ at most once, then $w(Q) \geq w(T[x, y])$.
2. If $Q$ is an $(x, y)$-path and $Q'$ is an $(x', y')$-path vertex-disjoint from $Q$, then $w(Q) + w(Q') \geq w(T[x, y] \setminus T[x', y']) + w(T[x', y'] \setminus T[x, y])$.
3.2 Solution Structure on Negative Trees

We first observe a simple property of minimum-weight solutions.

Definition 4 (Locally cheapest path pairs). Let $s_1, s_2, t_1, t_2$ be vertices in $G$, and let $P_1$ and $P_2$ be two permissively disjoint ($\{s_1, s_2\}, \{t_1, t_2\}$)-paths. A path $T[u,v]$ in some $T \in \mathcal{T}$ is called a shortcut for $P_1$ and $P_2$, if $u$ and $v$ both appear on the same path, either $P_1$ or $P_2$, and there is no inner vertex or edge of $P$ and $P'$, respectively, that has the given property.

The idea behind this concept is the following. Suppose that $P_1$ and $P_2$ are permissively disjoint ($\{s_1, s_2\}, \{t_1, t_2\}$)-paths, and $T[u,v]$ is a shortcut for $P_1$ and $P_2$. Suppose that $u$ and $v$ both lie on $P_i$ (for some $i \in [2]$), and let $P'_i$ be the path obtained by replacing $P_i[u,v]$ with $T[u,v]$; we refer to this operation as amending the shortcut $T[u,v]$. Then $P'_i$ is also permissively disjoint from $P_{3-i}$ and, since Lemma 2 implies $w(P_i[u,v]) \geq -w(T[u,v]) > 0$, has weight less than $w(P_i)$. Hence, we have the following observation.

Observation 5. Let $P_1$ and $P_2$ be two permissively disjoint ($\{s_1, s_2\}, \{t_1, t_2\}$)-paths admitting a shortcut $T[z,z']$. Suppose that $z$ and $z'$ are on the path, say, $P_1$. Let $P'_i$ be the path obtained by amending $T[z,z']$ on $P_1$. Then $P'_i$ and $P_2$ are permissively disjoint ($\{s_1, s_2\}, \{t_1, t_2\}$)-paths and $w(P'_i) < w(P_i)$.

Corollary 6. Any minimum-weight solution for $(G, w, s, t)$ is a pair of locally cheapest paths.

For convenience, for any $(s,t)$-path $P$ and vertices $u,v \in V(P)$ we say that $u$ precedes $v$ on $P$, or equivalently, $v$ follows $u$ on $P$, if $u$ lies on $P[s,v]$. When defining a vertex as the “first” (or “last”) vertex with some property on $P$ or on a subpath $P'$ of $P$ then, unless otherwise stated, we mean the vertex on $P$ or on $P'$ that is closest to $s$ (or farthest from $s$, respectively) that has the given property.

The following lemma shows that if two paths in a minimum-weight solution both use negative trees $T$ and $T'$ for some $T, T' \in \mathcal{T}$ then, roughly speaking, they must traverse $T$ and $T'$ in the same order; otherwise one would be able to replace the subpaths of the solution running between $T$ and $T'$ by two paths, one within $T$ and one within $T'$, of smaller weight.

Lemma 7. Let $P_1$ and $P_2$ be two openly disjoint $(s,t)$-paths of minimum total weight, and let $T$ and $T'$ be distinct trees in $\mathcal{T}$. Suppose that $v_1, v_2, v'_1$ and $v'_2$ are vertices such that $v_i \in V(T) \cap V(P_i)$ and $v'_i \in V(T') \cap V(P_i)$ for $i \in [2]$, with $v_1$ preceding $v'_1$ on $P_1$. Then $v_2$ precedes $v'_2$ on $P_2$.

The following lemma is a consequence of Corollary 6 and Lemma 7, and considers a situation when one of the paths in an optimal solution visits a negative tree $T \in \mathcal{T}$ at least twice, and visits some $T' \in \mathcal{T} \setminus \{T\}$ in between.

Lemma 8. Let $P_1$ and $P_2$ be two openly disjoint $(s,t)$-paths of minimum total weight, and let $T$ and $T'$ be distinct trees in $\mathcal{T}$. Suppose that $v_1, v'_2, v_2$ and $v_3$ are vertices appearing in this order on $P_1$ when traversed from $s$ to $t$, and suppose $v_1, v_3 \in V(T)$ while $v'_2 \in V(T')$. Then

- $V(P_2) \cap V(T) \neq \emptyset$;
- $V(P_2) \cap V(T') = \emptyset$;
- no vertex of $V(P_1) \cap V(T')$ precedes $v_1$ or follows $v_3$ on $P_1$.

We say that two paths $P_1$ and $P_2$ are in contact at $T$, if there is a tree $T \in \mathcal{T}$ and two distinct vertices $v_1$ and $v_2$ in $T$ such that $v_1$ lies on $P_1$, and $v_2$ lies on $P_2$. Lemmas 7 and 8 imply the following fact that will enable us to use recursion in our algorithm to find solutions that consist of two paths in contact.
Shortest Two Disjoint Paths in Conservative Graphs

Lemma 9. Let \( P_1 \) and \( P_2 \) be two openly disjoint \((s, t)\)-paths of minimum total weight, and assume that they are in contact at some tree \( T \in \mathcal{T} \). For \( i \in [2] \), let \( a_i \) and \( b_i \) denote the first and last vertices of \( P_i \) on \( T \) when traversed from \( s \) to \( t \). Then we can partition \( \mathcal{T} \setminus \{T\} \) into \( (\mathcal{T}_s, \mathcal{T}_0, \mathcal{T}_t) \) such that for each \( T' \in \mathcal{T} \setminus \{T\} \) and \( i \in [2] \) it holds that

(i) if \( P_i \), then \( T' \in \mathcal{T}_s \)
(ii) if \( P_i \), then \( T' \in \mathcal{T}_t \)
(iii) if \( P_i \), then \( T' \in \mathcal{T}_0 \).

Given a solution \((P_1, P_2)\) whose paths are in contact at some tree \( T \in \mathcal{T} \), a partition of \( \mathcal{T} \setminus \{T\} \) is \( T \)-valid with respect to \((P_1, P_2)\), if it satisfies the conditions of Lemma 9.

Combining Path Pairs

The following lemma will be a crucial ingredient in our algorithm, as it enables us to combine “partial solutions” without violating the requirement of vertex-disjointness.

Lemma 10. Let \( p_1, p_2, q_1, q_2 \) be vertices in \( G \), and let \( T \in \mathcal{T} \) contain vertices \( v_1 \) and \( v_2 \) with \( v_1 \neq v_2 \). Let \( P_1 \) and \( P_2 \) be two permissively disjoint \((\{p_1, p_2\}, \{v_1, v_2\})\)-paths in \( G \), and let \( Q_1 \) and \( Q_2 \) be two permissively disjoint \((\{v_1, v_2\}, \{q_1, q_2\})\)-paths in \( G \) that are locally cheapest. Assume also that we can partition \( \mathcal{T} \) into two sets \( \mathcal{T}_1 \) and \( \mathcal{T}_2 \) with \( T \in \mathcal{T}_2 \) such that

(i) \( V(T) \cap V(P_1 \cup P_2) = \{v_1, v_2\} \), and
(ii) \( P_1 \cup P_2 \) contains no edge of \( E(T_2) \), and \( Q_1 \cup Q_2 \) contains no edge of \( E(T_1) \).

Then we can find in linear time two permissively disjoint \((\{p_1, p_2\}, \{q_1, q_2\})\)-paths \( S_1 \) and \( S_2 \) in \( G \) such that \( w(S_1) + w(S_2) \leq w(P_1) + w(P_2) + w(Q_1) + w(Q_2) \).

We remark that Lemma 10 heavily relies on the condition that \( v_1 \) and \( v_2 \) are both on \( T \); to construct the desired paths \( S_1 \) and \( S_2 \), we not only use the path pairs \((P_1, P_2)\) and \((Q_1, Q_2)\) but, if necessary, remove certain subpaths from them and stitch together the remainder with a path running within \( T \).

Polynomial-Time Algorithm for Constant \(|T|\)

This section contains the algorithm proving our main result, Theorem 1. Let \((G, w, s, t)\) be our instance of Shortest Two Disjoint Paths with input graph \( G = (V, E) \), and assume that the set \( E^- \) of negative edges spans \( c \) trees in \( G \) for some constant \( c \). We present a polynomial-time algorithm that computes a solution for \((G, w, s, t)\) with minimum total weight, or correctly concludes that no solution exists for \((G, w, s, t)\). The running time of our algorithm is \( O(n^{2c+1}) \) where \( n = |V| \), so in the language of parameterized complexity, our algorithm is in XP with respect to the parameter \( c \).

In Section 4.1 we present the main ideas and definitions necessary for our algorithm, and provide its high-level description together with some further details. We will distinguish between so-called separable and non-separable solutions. Finding an optimal and separable solution will be relatively easy, requiring extensive guessing but only standard techniques for computing minimum-cost flows. By contrast, finding an optimal but non-separable solution is much more difficult. Therefore, in Section 4.2 we collect useful properties of optimal, non-separable solutions. These observations form the basis for an important subroutine necessary for finding optimal, non-separable solutions; this subroutine is presented in Section 4.3.
4.1 The Algorithm

We distinguish between two types of solutions for our instance \((G, w, s, t)\) of Shortest Two Disjoint Paths.

**Definition 11 (Separable solution).** Let \((P_1, P_2)\) be a solution for \((G, w, s, t)\). We say that \(P_1\) and \(P_2\) are separable, if either

- they are not in contact, i.e., there is no tree \(T \in \mathcal{T}\) that shares distinct vertices with both \(P_1\) and \(P_2\), or
- there is a unique tree \(T \in \mathcal{T}\) such that \(P_1\) and \(P_2\) are in contact at \(T\), but the intersection of \(T\) with both \(P_1\) and \(P_2\) is a path, possibly containing only a single vertex; otherwise they are non-separable.

In Section 4.1.1 we show how to find an optimal, separable solution, whenever such a solution exists for \((G, w, s, t)\). Section 4.1.2 deals with the case when we need to find an optimal, non-separable solution. Our algorithm for the latter case is more involved, and relies on a subroutine that is based on dynamic programming and is developed throughout Sections 4.2 and 4.3. The existence of this subroutine is stated in Corollary 33.

4.1.1 Finding Separable Solutions

Suppose that \((P_1, P_2)\) is a minimum-weight solution for \((G, w, s, t)\) that is separable. The following definition establishes conditions when we are able to apply a simple strategy for finding a minimum-weight solution using well-known flow techniques.

**Definition 12 (Strongly separable solution).** Let \((P_1, P_2)\) be a solution for \((G, w, s, t)\). We say that \(P_1\) and \(P_2\) are strongly separable, if they are separable and they are either not in contact at any tree of \(\mathcal{T}\), or they are contact at a tree of \(\mathcal{T}\) that contains \(s\) or \(t\).

Suppose that \(P_1\) and \(P_2\) are either not in contact, or they are in contact at a tree of \(\mathcal{T}\) that contains \(s\) or \(t\). Let \(\mathcal{T}^{3s,t}\) denote the set of all trees in \(\mathcal{T}\) that contain neither \(s\) nor \(t\). For each \(T \in \mathcal{T}^{3s,t}\) that shares a vertex with \(P_i\) for some \(i \in [2]\), we define \(a_T\) as the first vertex on \(P_1\) (when traversed from \(s\) to \(t\)) that is contained in \(T\); note that \(T\) cannot share vertices with both \(P_1\) and \(P_2\) as they are strongly separable, so \(P_1\) is uniquely defined.

Our approach is the following: we guess the vertex \(a_T\) for each \(T \in \mathcal{T}^{3s,t}\), and then compute a minimum-cost flow in an appropriately defined network. More precisely, for each possible choice of vertices \(Z = \{z_T \in V(T) : T \in \mathcal{T}^{3s,t}\}\), we build a network \(N_Z\) as follows.

**Definition 13 (Flow network \(N_Z\) for strongly separable solutions).** Given a set \(Z \subseteq V\) such that \(Z \cap V(T) = \{z_T\}\) for each \(T \in \mathcal{T}^{3s,t}\), we create \(N_Z\) as follows. We direct each non-negative edge in \(G\) in both directions. Then for each \(T \in \mathcal{T}^{3s,t}\), we direct the edges of \(T\) away from \(z_T\).\(^3\) If some \(T \in \mathcal{T}\) contains \(s\), then we direct all edges of \(T\) away from \(s\); similarly, if some \(T \in \mathcal{T}\) contains \(t\), then we direct all edges of \(T\) towards \(t\). We assign a capacity of 1 to each arc and to each vertex\(^4\) in the network except for \(s\) and \(t\), and we retain the cost function \(w\) (meaning that we define \(w(\vec{e})\) as \(w(e)\) for any arc \(\vec{e}\) obtained by directing some edge \(e\)). We let \(s\) and \(t\) be the source and the sink in \(N_Z\), respectively.

---

\(^3\) Directing a tree \(T\) away from a vertex \(z \in V(T)\) means that an edge \(uv\) in \(T\) becomes an arc \((u, v)\) if and only if \(T[z, u]\) has fewer edges than \(T[z, v]\); directing \(T\) towards \(z\) is defined analogously.

\(^4\) The standard network flow model can be adjusted by well-known techniques to allow for vertex capacities.
Lemma 14. If there exists a strongly separable solution for \((G, w, s, t)\) with weight \(k\), then there exists a flow of value 2 having cost \(k\) in the network \(N_Z\) for some choice of \(Z \subseteq V\) containing exactly one vertex from each tree in \(T^\#\). Conversely, a flow of value 2 and cost \(k\) in the constructed network \(N_Z\) for some \(Z\) yields a solution for \((G, w, s, t)\) with weight at most \(k\).

Next, we show how to deal with the case when a minimum-weight solution \((P_1, P_2)\) is separable, but not strongly separable; then there is a unique tree \(T \in \mathcal{T}\) at which \(P_1\) and \(P_2\) are in contact. In such a case, we can simply delete an edge from \(T\) in a way that paths \(P_1\) and \(P_2\) cease to be in contact in the resulting instance. This way, we can reduce our problem to the case when there is a strongly separable optimal solution; note, however, that the number of trees spanned by the negative edges (our parameter \(c\)) increases by 1.

Lemma 15. If there exists a separable, but not strongly separable solution for \((G, w, s, t)\) with weight \(k\), then there exists an edge \(e \in E^-\) such that setting \(G' = G - e\) and \(E' = E \setminus \{e\}\), the instance \((G', w|_{E'}, s, t)\) admits a strongly separable solution with weight at most \(k\). Conversely, a solution for \((G', w|_{E'}, s, t)\) is also a solution for \((G, w, s, t)\) with the same weight.

Thanks to Lemmas 14 and 15, if there exists a minimum-weight solution for \((G, w, s, t)\) that is separable, then we can find some minimum-weight solution using standard algorithms for computing minimum-cost flows. In Section 4.1.2 we explain how we can find a minimum-weight non-separable solution for \((G, w, s, t)\).

### 4.1.2 Finding Non-separable Solutions

To find a minimum-weight solution for \((G, w, s, t)\) that is not separable, we need a more involved approach. We now provide a high-level presentation of our algorithm for finding a non-separable solution of minimum weight. We remark that Algorithm STDP contains a pseudocode; however, we believe that it is best to read the following description first.

**Step 1.** We guess certain properties of a minimum-weight non-separable solution \((P_1, P_2)\): First, we guess a tree \(T \in \mathcal{T}\) that shares distinct vertices both with \(P_1\) and \(P_2\), i.e., a tree at which \(P_1\) and \(P_2\) are in contact. Second, we guess a partition \((T_s, T_0, T_t)\) of \(T \setminus \{T\}\) that is \(T\)-valid with respect to \((P_1, P_2)\).

**Step 2.** If \(T_s \neq \emptyset\), then we guess the first vertex of \(P_1\) and of \(P_2\) contained in \(V(T)\), denoted by \(a_1\) and \(a_2\), respectively. We use recursion to compute two permissively disjoint \((s, \{a_1, a_2\})\)-paths using only the negative trees in \(T_s\) and to compute two permissively disjoint \((\{a_1, a_2\}, t)\)-paths using only the negative trees in \(T \setminus T_s\). Observe that by \(T_s \neq \emptyset\) and \(T \in \mathcal{T} \setminus T_s\), we search for these paths in graphs that contain only a strict subset of the negative trees in \(T\). Thus, our parameter \(c\) strictly decreases in both constructed sub-instances. We combine the obtained pairs of paths into a solution by using Lemma 10.

We proceed in a similar fashion when \(T_t \neq \emptyset\).

**Step 3.** If \(T_s = T_t = \emptyset\), then for both \(i \in [2]\) we guess the first and last vertex of \(P_i\) contained in \(V(T)\), denoted by \(a_i\) and \(b_i\), respectively. We apply standard flow techniques to compute two \((s, \{a_1, a_2\})\)-paths and two \((\{b_1, b_2\}, t)\)-paths with no inner vertices in \(V(T)\) that are pairwise permissively disjoint. Then, we apply the polynomial-time algorithm we devise for computing a pair of permissively disjoint \((\{a_1, a_2\}, \{b_1, b_2\})\)-paths in \(G\). This algorithm is the cornerstone of our method, and is based on important structural observations that allow for efficient dynamic programming. We combine the obtained pairs of paths into a solution by applying Lemma 10.

**Step 4.** We output a solution of minimum weight among all solutions found in Steps 2 and 3.

Let us now provide more details about these steps; see also Algorithm STDP.
The same holds when substituting compute in linear time a solution for a with an edge of weight w is.

We obtain two permissively disjoint instances (by deleting the vertex a* and connecting it to both a1 and a2) as our two terminals.

To apply recursion in Step 2 when Ts ≠ ∅, we guess two distinct vertices a1 and a2 in T, and define the following sub-instances.

**Definition 16 (Sub-instances for Ts ≠ ∅).** For some ∅ ≠ Ts ⊆ T and distinct vertices a1 and a2, we create a graph Gs by adding a new vertex a* and connecting it to both a1 and a2 with an edge of weight w*a = |w(T(a1, a2))|/2. We create instances I1 and I2 as follows:

- to get I1, we delete E(T \ Ts) and V(T \ Ts) \ {a1, a2} from Gs, and designate s and a* as our two terminals;
- to get I2, we delete V(Ts) from Gs, and designate a* and t as our two terminals.

We use recursion to solve Shortest Two Disjoint Paths on sub-instances I1 and I2. Note that conservativeness is maintained for both I1 and I2, due to our choice of wa, and statement (1) of Lemma 3. If both sub-instances admit solutions, we obtain two permissively disjoint (s, {a1, a2})-paths Q1 and Q2 from our solution to I1 by deleting the vertex a*. Similarly, we obtain two permissively disjoint ((a1, a2), t)-paths Q1 and Q2 from our solution to I2 by deleting the vertex a*.

If Ts ≠ ∅, then we proceed similarly: after guessing two distinct vertices b1 and b2 in T, we use a construction analogous to Definition 16.

**Definition 17 (Sub-instances for Ts ≠ ∅).** For some ∅ ≠ Ts ⊆ T and distinct vertices b1 and b2, we create a graph Gi by adding a new vertex b* and connecting it to both b1 and b2 with an edge of weight wb, = |w(T(b1, b2))|/2. We create instances I1 and I2 as follows:

- to get I1, we delete V(Ti) from Gi, and designate s and b* as our two terminals;
- to get I2, we delete E(T \ Ti) and V(T \ Ti) \ {b1, b2} from Gi, and designate b* and t as our two terminals.

We use recursion to solve Shortest Two Disjoint Paths on sub-instances I1 and I2; again, conservativeness is ensured for I1 and I2 by our choice of wb, and statement (1) of Lemma 3. If both sub-instances admit solutions, we obtain two permissively disjoint (s, {b1, b2})-paths Q1 and Q2 from our solution to I1 by deleting the vertex b*. Similarly, we obtain two permissively disjoint ((b1, b2), t)-paths Q1 and Q2 from our solution to I2 by deleting the vertex b*. Again, we use Lemma 10 to create a solution for our original instance (G, w, s, t) using paths Q1, Q2, Q1, and Q2.

We state the correctness of Step 2 in the following lemma:

**Lemma 18.** Suppose that (P1, P2) is a minimum-weight solution for (G, w, s, t) such that

- P1 and P2 are in contact at some T ∈ T,
- ai and bi are the first and last vertices of Pi contained in T, respectively, for i ∈ [2], and
- (Ts, T0, Ts) is a T-valid partition w.r.t. (P1, P2).

Then instances I1 and I2 admit solutions whose total weight (summed over all four paths) is w(P1) + w(P2) + 4wa. Furthermore, given a solution Si, for i ∈ [2], we can compute in linear time a solution for (G, w, s, t) of weight at most w(Si) + w(S2) − 4wa.

The same holds when substituting I1, I2, and wa, with I1, I2, and wa in these claims.
Algorithm STDP Solving Shortest Two Disjoint Paths with conservative weights.

Input: An instance $(G,w,s,t)$ where $w$ is conservative on $G$.

Output: A solution for $(G,w,s,t)$ with minimum weight, or $\emptyset$ if no solution exist.

1: Let $S = \emptyset$.
2: for all $E' \subseteq E$ such that $E \setminus E' \subseteq E^-$ and $|E \setminus E'| \leq 1$ do $\triangleright$ Separable solutions.
   3: Create the instance $(G[E'],w|_{E'},s,t)$.
   4: Let $T^s,t = \{T : T$ is a maximal tree in $G[E' \cap E^-]$ with $s,t \notin V(T)\}$.
   5: for all $Z \subseteq V$ such that $|Z \cap V(T)| = 1$ for each $T \in T^s,t$ do
      6: Create the network $N_Z$ from instance $(G[E'],w|_{E'},s,t)$.$\triangleright$ Use Def. 13.
      7: if $\exists$ a flow $f$ of value 2 in $N_Z$ then
         8: Compute a minimum-cost flow $f$ of value 2 in $N_Z$.
         9: Construct a solution $(S_1,S_2)$ from $f$ using Lemma 14.
      10: $S \leftarrow (S_1,S_2)$.$\triangleright$ Non-separable solutions.
   11: for all $T \in T$ do
      12: for all partitions $(T_0,T_1,T_i)$ of $T \setminus \{T\}$ do
         13: if $T_i \neq \emptyset$ then
            14: for all $a_1,a_2 \in V(T)$ with $a_1 \neq a_2$ do
               15: Create sub-instances $I^1_i$ and $I^2_i$.$\triangleright$ Use Def. 16.
               16: Compute $(Q^1_i,Q^2_i) = \text{STDP}(I^1_i)$.
               17: Compute $(Q^1_i,Q^2_i) = \text{STDP}(I^2_i)$.
            18: if $(Q^1_i,Q^2_i) \neq \emptyset$ and $(Q^1_i,Q^2_i) \neq \emptyset$ then
               19: Create a solution $(S_1,S_2)$ from $(Q^1_i,Q^2_i)$ and $(Q^1_i,Q^2_i)$ using Lemma 18.
               20: $S \leftarrow (S_1,S_2)$.
            21: else if $T_i \neq \emptyset$ then
               22: for all $b_1,b_2 \in V(T)$ with $b_1 \neq b_2$ do
                  23: Create sub-instances $I^1_i$ and $I^2_i$.$\triangleright$ Use Def. 17.
                  24: Compute $(Q^1_i,Q^2_i) = \text{STDP}(I^1_i)$.
                  25: Compute $(Q^1_i,Q^2_i) = \text{STDP}(I^2_i)$.
               26: if $(Q^1_i,Q^2_i) \neq \emptyset$ and $(Q^1_i,Q^2_i) \neq \emptyset$ then
                  27: Create a solution $(S_1,S_2)$ from $(Q^1_i,Q^2_i)$ and $(Q^1_i,Q^2_i)$ using Lemma 18.
                  28: $S \leftarrow (S_1,S_2)$.$\triangleright$ $T_0 = T_i = \emptyset$.
               29: else
                  30: for all $a_1,a_2,b_1,b_2 \in V(T)$ that constitute a reasonable guess do
                     31: if $\exists$ a flow $f$ of value 4 in $N_{(a_1,b_1,a_2,b_2)}$ then $\triangleright$ Use Def. 19.
                        32: if $\exists$ two permissively disjoint $(\{a_1,a_2\},\{b_1,b_2\})$-paths in $G$ then
                           33: Compute a minimum-cost flow $f$ of value 4 in $N_{(a_1,b_1,a_2,b_2)}$.
                           34: Compute permissively disjoint $(\{a_1,a_2\},\{b_1,b_2\})$-paths $Q_1$ and $Q_2$.
                           35: $\triangleright$ Use Corollary 33 in Section 4.3
                           36: Construct a solution $(S_1,S_2)$ from $f$, $Q_1$, and $Q_2$ using Lemma 20.
                           37: $S \leftarrow (S_1,S_2)$.
                  38: if $S = \emptyset$ then return $\emptyset$.
               39: else Let $S^*$ be the cheapest pair among those in $S$, and return $S^*$.
Step 3: Applying flow techniques and dynamic programming

We now describe Step 3 in more detail, which concerns the case when $T_s = T_t = \emptyset$.

First, we guess vertices $a_1, b_1, a_2,$ and $b_2$: the intended meaning of these vertices is that $a_i$ and $b_i$ are the first and last vertices of $P_i$ contained in $V(T)$, for both $i \in [2]$. We only consider guesses that are reasonable, meaning that they satisfy the following conditions:

- if $s \in V(T)$, then $s = a_1 = a_2$, otherwise $a_1 \neq a_2$;
- if $t \in V(T)$, then $t = b_1 = b_2$, otherwise $b_1 \neq b_2$;
- $T[a_1, b_1]$ and $T[a_2, b_2]$ share at least one edge.

Then we compute four paths from $\{s, t\}$ to $\{a_1, b_1, a_2, b_2\}$ in the graph $G - E(T) - (V(T) \setminus \{a_1, a_2, b_1, b_2\})$ with minimum weight such that two paths have $s$ as an endpoint, the other two have $t$ as an endpoint, and no other vertex appears on more than one path. To this end, we define the following network and compute a minimum-cost flow of value 4 in it.

Definition 19 (Flow network $N_{(a_1, b_1, a_2, b_2)}$ for non-separable solutions.). Given vertices $a_1, b_1, a_2,$ and $b_2$, we create $N_{(a_1, b_1, a_2, b_2)}$ as follows. First, we delete all edges in $E^- \cup T$ and all vertices in $V(T) \setminus \{a_1, a_2, b_1, b_2\}$ from $G$, and direct each edge in $G$ in both directions. We then add new vertices $s^*$ and $t^*$, along with arcs $(s^*, s)$ and $(s^*, t)$ of capacity 2, and arcs $(a_i, t^*)$ and $(b_i, t^*)$ for $i = 1, 2$ with capacity 1.\footnote{In the degenerate case when $s = a_1 = a_2$ or $t = b_1 = b_2$ this yields two parallel arcs from $s$ or $t$ to $t^*$.} We assign capacity 1 to all other arcs, and also to each vertex of $V(G) \setminus \{s, t\}$. All newly added arcs will have cost 0, otherwise we retain the cost function $w$. We let $s^*$ and $t^*$ be the source and the sink in $N_{(a_1, b_1, a_2, b_2)}$, respectively.

Next, we compute two $\{(a_1, a_2), (b_1, b_2)\}$-paths $Q_1$ and $Q_2$ in $G$ with minimum total weight that are permissively disjoint. A polynomial-time computation for this problem, building on structural observations from Section 4.2, is provided in Section 4.3 (see Corollary 33). Finally, we apply Lemma 10 (in fact, twice) to obtain a solution to our instance $(G, w, s, t)$ based on a minimum-cost flow of value 4 in $N_{(a_1, b_1, a_2, b_2)}$ and paths $Q_1$ and $Q_2$. We finish this section with Lemma 20, stating the correctness of Step 3.

Lemma 20. Suppose that $(P_1, P_2)$ is a minimum-weight solution for $(G, w, s, t)$ such that

- $P_1$ and $P_2$ are non-separable, and in contact at some $T \in \mathcal{T}$,
- $a_i$ and $b_i$ are the first and last vertices of $P_i$ contained in $T$, respectively, for $i \in [2]$, and
- $(\emptyset, T \setminus \{T\}, \emptyset)$ is a T-valid partition w.r.t. $(P_1, P_2)$.

Let $Q_1$ and $Q_2$ be two permissively disjoint $\{(a_1, a_2), (b_1, b_2)\}$-paths in $G$ with minimum total weight. Then the following holds:

If $w^*$ is the minimum cost of a flow of value 4 in the network $N_{(a_1, b_1, a_2, b_2)}$, then $w^* + w(Q_1) + w(Q_2) \leq w(P_1) + w(P_2)$. Conversely, given a flow of value 4 in the network $N_{(a_1, b_1, a_2, b_2)}$ with cost $w^*$, together with paths $Q_1$ and $Q_2$, we can find a solution for $(G, w, s, t)$ with cost at most $w^* + w(Q_1) + w(Q_2)$ in linear time.

4.2 Properties of a Non-separable Solution

Let us now turn our attention to the subroutine lying at the heart of our algorithm for Shortest Two Disjoint Paths: an algorithm that, given two source terminals and two sink terminals on some tree $T \in \mathcal{T}$, computes two permissively disjoint paths from the two source terminals to the two sink terminals, with minimum total weight. It is straightforward to see that any non-separable solution whose paths are in contact at $T$ contains such a pair...
of paths. Therefore, as described in Section 4.1.2, finding such paths is a necessary step to computing an optimal, non-separable solution for our instance \((G, w, s, t)\). This section contains observations about the properties of such paths.

Let us now formalize our setting. Let \(a_1, a_2, b_1,\) and \(b_2\) be vertices on a fixed tree \(T \in \mathcal{T}\) such that \(T[a_1, b_1]\) and \(T[a_2, b_2]\) intersect in a path \(X\) (with at least one edge), with one component of \(T \setminus X\) containing \(a_1\) and \(a_2\), and the other containing \(b_1\) and \(b_2\). Let the vertices on \(X\) be \(x_1, \ldots, x_r\) with \(x_1\) being the closest to \(a_1\) and \(a_2\). We will use the notation \(A_i = T[a_i, x_i]\) and \(B_i = T[b_i, x_i]\) for each \(i \in [2]\). For each \(i \in [r]\), let \(T_i\) be the maximal subtree of \(T\) containing \(x_i\) but no other vertex of \(X\). We also define \(T_{(i,j)} = \bigcup_{h \leq h \leq j} T_h\) for some \(i\) and \(j\) with \(1 \leq i \leq j \leq r\).

For convenience, for any path \(Q\) that has \(a_i \in \{a_1, a_2\}\) as its endpoint, we will say that \(Q\) starts at \(a_i\) and ends at its other endpoint. Accordingly, for vertices \(u, v \in V(Q)\) we say that \(u\) precedes \(v\) on \(Q\), or equivalently, \(v\) follows \(u\) on \(Q\), if \(u\) lies on \(Q[a_i, v]\). When defining a vertex as the “first” (or “last”) vertex with some property on \(Q\) or on a subpath \(Q'\) of \(Q\) then, unless otherwise stated, we mean the vertex on \(Q\) or on \(Q'\) that is closest to \(a_i\) (or farthest from \(a_i\), respectively) that has the given property.

Using Lemma 2 and (an extended version of) Lemma 3, we can establish the following properties of a non-separable minimum-weight solution.

\begin{definition}[X-monotone path] A path \(P\) starting at \(a_1\) or \(a_2\) is X-monotone if for any vertices \(u_1\) and \(u_2\) on \(Q\) such that \(u_1 \in V(T_{j_1})\) and \(u_2 \in V(T_{j_2})\) for some \(j_1 < j_2\) it holds that \(u_1\) precedes \(u_2\) on \(Q\).
\end{definition}

\begin{definition}[Plain path] A path \(P\) is plain, if whenever \(Q\) contains some \(x_i \in V(X)\), then the vertices of \(Q\) in \(T_i\) induce a path in \(T_i\). In other words, if vertices \(x_j \in V(X)\) and \(u \in V(T_j)\) both appear on \(Q\), then \(T[u, x_j] \subseteq Q\).
\end{definition}

\begin{lemma}
Let \(Q_1\) and \(Q_2\) be two permissively disjoint \((\{a_1, a_2\}, \{b_1, b_2\})\)-paths in \(G\) with minimum total weight. Then both \(Q_1\) and \(Q_2\) are X-monotone and plain.
\end{lemma}

The following observation summarizes our understanding on how an optimal solution uses paths \(A_1, A_2, B_1,\) and \(B_2\).

\begin{lemma}
If \(Q_1\) and \(Q_2\) are two permissively disjoint \((\{a_1, a_2\}, \{b_1, b_2\})\)-paths that are locally cheapest and also plain, then one of them contains \(A_1\) or \(A_2\), and one of them contains \(B_1\) or \(B_2\).
\end{lemma}

### 4.3 Computing Partial Solutions

In this section we design a dynamic programming algorithm that computes two permissively disjoint \((\{a_1, a_2\}, \{b_1, b_2\})\)-paths of minimum total weight (we keep all definitions introduced in Section 4.2, including our assumptions on vertices \(a_1, b_1, a_2,\) and \(b_2\)). In Section 4.2 we have established that two permissively disjoint \((\{a_1, a_2\}, \{b_1, b_2\})\)-paths of minimum total weight are necessarily X-monotone, plain, and they form a locally cheapest pair. A natural approach would be to require these same properties from a partial solution that we aim to compute. However, it turns out that the property of X-monotonicity is quite hard to ensure when building subpaths of a solution. The following relaxed version of monotonicity can be satisfied much easier, and still suffices for our purposes:

\begin{definition}[Quasi-monotone path] A path \(P\) starting at \(a_1\) or \(a_2\) is quasi-monotone, if the following holds: if \(x_i \in V(P)\) for some \(i \in [r]\), then all vertices in \(\bigcup_{h \in [i-1]} V(T_h) \cap V(P)\) precede \(x_i\) on \(P\), and all vertices in \(\bigcup_{h \in [r]\setminus[i]} V(T_h) \cap V(P)\) follow \(x_i\) on \(P\).
\end{definition}
Definition 26 (Well-formed path pair). Two paths $P_1$ and $P_2$ form a well-formed pair, if they are locally cheapest, and both are plain and quasi-monotone.

We are now ready to define partial solutions, the central notion that our dynamic programming algorithm relies on.

Definition 27 (Partial solution). Given vertices $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$ and a set $\tau \subseteq T \setminus \{T\}$, two paths $Q_1$ and $Q_2$ form a partial solution $(Q_1, Q_2)$ for $(u, v, \tau)$, if

(a) $Q_1$ and $Q_2$ are permissively disjoint $\langle \{a_1, a_2\}, \{u, v\}\rangle$-paths;
(b) $Q_1$ and $Q_2$ are a well-formed pair;
(c) $Q_1$ ends with the subpath $T[x_i, u]$;
(d) $V(T_{i+1,r_i}) \cap V(Q_2) \subseteq \{v\}$;
(e) if $Q_1 \cup Q_2$ contains a vertex of some $T' \in T \setminus \{T\}$, then $T' \in \tau$;
(f) there exists no tree $T' \in T \setminus \{T\}$ such that $Q_1$ and $Q_2$ are in contact at $T'$.

We will say that the vertices of $V(T \setminus (\tau \cup \{T\}))$ are forbidden for $(\tau, T)$; then condition (e) asks for $Q_1 \cup Q_2$ not to contain vertices forbidden for $(\tau, T)$.

Before turning our attention to the problem of computing partial solutions, let us first show how partial solutions enable us to find two permissively disjoint $\langle \{a_1, a_2\}, \{b_1, b_2\}\rangle$-paths.

Lemma 28. Paths $P_1$ and $P_2$ are permissively disjoint $\langle \{a_1, a_2\}, \{b_1, b_2\}\rangle$-paths of minimum weight in $G$ if and only if they form a partial solution for $\langle b_1, b_{3-h}, T \setminus \{T\} \rangle$ of minimum weight for some $h \in [2]$.

We now present our approach for computing partial solutions using dynamic programming.

Computing partial solutions: high-level view

For each $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$, and each $\tau \subseteq T \setminus \{T\}$, we are going to compute a partial solution for $(u, v, \tau)$ of minimum weight, denoted by $F(u, v, \tau)$, using dynamic programming; if there exists no partial solution for $(u, v, \tau)$, we set $F(u, v, \tau) = \emptyset$.

To apply dynamic programming, we fix an ordering $\prec$ over $V(T)$ fulfilling the condition that for each $i' < i$, $u \in V(T_i)$ and $u' \in V(T_{i'})$ we have $u' \prec u$. We compute the values $F(u, v, \tau)$ based on the ordering $\prec$ in the sense that $F(u', v', \tau')$ is computed before $F(u, v, \tau)$ whenever $u' \prec u$. This computation is performed by Algorithm PARTSOL which determines a partial solution $F(u, v, \tau)$ based on partial solutions already computed.

To compute $F(u, v, \tau)$ in a recursive manner, we use an observation that either the partial solution has a fairly simple structure, or it strictly contains a partial solution for $(u', v', \tau')$ for some vertices $u'$ and $v'$ with $u' \in V(T_{i'})$ and $i' < i$, and some set $\tau' \subseteq \tau$. We can thus try all possible values for $u'$, $v'$ and $\tau'$, and use the partial solution $(Q_1', Q_2')$ we have already computed and stored in $F(u', v', \tau')$. To obtain a partial solution for $(u, v, \tau)$ based on $Q_1'$ and $Q_2'$, we append paths to $Q_1'$ and to $Q_2'$ so that they fulfill the requirements of Definition 27—most importantly, that $Q_1$ ends with $T[x_i, u]$, that $Q_2$ ends at $v$, and that $Q_1 \cup Q_2$ contains no vertex of $V(T \setminus (\tau \cup \{T\}))$. To this end, we create a path $P_1 = Q_1' \cup T[v', u]$ and a path $P_2 = Q_2' \cup R$ where $R$ is a shortest $(u', v)$-path in a certain auxiliary graph. Essentially, we use the tree $T$ for getting from $v'$ to $u$, and we use the “remainder” of the graph for getting from $u'$ to $v$; note that we need to avoid the forbidden vertices and ensure condition (f) as well. The precise definition of the auxiliary subgraph of $G$ that we use for this purpose is provided in Definition 29. If the obtained path pair $(P_1, P_2)$ is indeed a partial solution for $(u, v, \tau)$, then we store it. After trying all possible values for $u'$, $v'$, and $\tau'$, we select a partial solution that has minimum weight among those we computed.
Definition 29 (Auxiliary graph). For some $T \in \mathcal{T}$, let $P \subseteq T$ be a path within $T$, let $u$ and $v$ be two vertices on $T$, and let $\tau \subseteq \mathcal{T} \setminus \{T\}$. Then the auxiliary graph $G(P, u, v, \tau)$ denotes the graph defined as

$$G(P, u, v, \tau) = G - \left( \bigcup \{V(T_h) : V(T_h) \cap V(P) = \emptyset\} \cup V(P) \setminus \{u, v\} \cup V(T \setminus (\tau \cup \{T\})) \right).$$

In other words, we obtain $G(P, u, v, \tau)$ from $G$ by deleting all trees $T_h$ that do not intersect $P$, and deleting $P$ itself as well, while taking care not to delete $u$ or $v$, and additionally deleting all vertices forbidden for $(\tau, T)$.

Working towards explaining the main ideas behind Algorithm PartSol, we start with two simple observations. The first one, stated by Lemma 30 below, essentially says that a path in a partial solution that uses a subtree $T_h$ of $T$ for some $h \in [\tau]$ should also go through the vertex $x_h$ whenever possible, that is, unless the other path uses $x_h$.

Lemma 30. Let $Q_1$ and $Q_2$ be two permissively disjoint, locally cheapest $\{(a_1, a_2), \{x, v\}\}$-paths for some $v \in V(T_j)$ where $1 \leq i \leq j \leq \tau$. Let $z \in V(T_h)$ for some $h \leq j$ such that $h < j$ or $x_h \in V(T[z, v])$. If $z \in V(Q_1 \cup Q_2)$, then $x_h \in V(Q_1 \cup Q_2)$.

As a consequence of Lemma 30, applied with $a_1$ taking the role of $z$ and $x_1$ taking the role of $x_h$, we get that every partial solution for some $(u, v, \tau)$ must contain $x_1$; using that both paths in a partial solution must be plain, we get the following fact.

Observation 31. Let $(Q_1, Q_2)$ be a partial solution for $(u, v, \tau)$ for vertices $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$ and for $\tau \subseteq \mathcal{T} \setminus \{T\}$. Then either $Q_1$ or $Q_2$ contains $A_1$ or $A_2$.

Let us now give some insight on Algorithm PartSol that computes a minimum-weight partial solution $(Q_1, Q_2)$ for $(u, v, \tau)$, if it exists, for vertices $u \in V(T_i)$ and $v \in V(T_j)$ with $i \leq j$ and trees $\tau \subseteq \mathcal{T} \setminus \{T\}$. It distinguishes between two cases based on whether $Q_2$ contains a vertex of $T[x_1, x_i]$ or not; see Figure 1 for an illustration. In both cases it constructs candidates for a partial solution, and then chooses one among these with minimum weight.

Case A: $Q_2$ does not contain any vertices from $T[x_1, x_i]$. In this case, due to Observation 31, we know that $Q_1$ contains $A_h$ for some $h \in [2]$; let us fix this value of $h$. Since $Q_1$ and $Q_2$ are locally cheapest and $Q_1$ ends with $T[x_1, u]$, we also know that $Q_1$ must contain $T[x_1, x_i]$. Therefore, we obtain $Q_1 = A_h \cup T[x_1, u]$. In this case, we can also prove that $Q_2$ is a shortest $(a_{3-h}, v, u)$-path in the auxiliary graph $G(A_h \cup T[x_1, u], a_{3-h}, v, u)$. Hence, Algorithm PartSol computes such a path $R$ and constructs the pair $(A_h \cup T[x_1, u], R)$ as a candidate for a partial solution for $(u, v, \tau)$.

Case B: $Q_2$ contains a vertex from $T[x_1, x_i]$. In this case, let $x_{i'}$ be the vertex on $T[x_1, x_{i-1}]$ closest to $x_1$ that appears on $Q_2$, and let $u'$ be the last vertex of $Q_2$ in $T_i'$; since $Q_2$ is plain, we know $T[x_{i'}, u'] \subseteq Q_2$. Let $x_{i'}'$ denote the vertex on $T[x_{i'}, x_i]$ closest to $x_{i'}$ that appears on $Q_1$; then $i' < j' \leq i$. As $Q_1$ and $Q_2$ are locally cheapest, $T[x_{i'}, x_i] \subseteq Q_1$ follows. Let $u'$ denote the first vertex of $Q_2$ in $T_{i'}$. Since $Q_1$ is plain, we know $T[u', x_{i'}] \subseteq Q_1$.

Define $Q_1 = Q_2 \setminus \{u', v\}$ and $Q_2 = Q_1 \setminus \{u', u\}$. Let also $\tau'$ denote those trees in $\mathcal{T} \setminus \{T\}$ that share a vertex with $Q_1 \cup Q_2$. We can then prove that $(Q_1, Q_2)$ is a partial solution for $(u', v', \tau')$; moreover, $Q_2[u', v]$ is a path in the auxiliary graph $G(T[u', v], u', v, \tau \setminus \tau')$. Thus, Algorithm PartSol takes a partial solution $(Q_1', Q_2')$ for $(u', v', \tau')$, already computed, and computes a shortest $(u', v')$-path $R$ in $G(T[u', v], u', v, \tau \setminus \tau')$. It then creates the path pair $(Q_2' \cup T[u', v], Q_1' \cup R)$ as a candidate for a partial solution for $(u, v, \tau)$.
Case B. The subpaths $Q_1$ and $Q_2$, respectively, form a partial solution for $(u', v', \tau')$ and are depicted in bold, with their endings marked by a parenthesis-shaped delimiter.

**Figure 1** Illustration for Algorithm PartSol. Edges within $T$ are depicted using solid lines, edges not in $T$ using dashed lines. Paths $Q_1$ and $Q_2$ are shown in **blue** and in **green**, respectively (see the online version for colored figures).

**Algorithm PartSol** Computes a partial solution $F(u, v, \tau)$ of minimum weight for $(u, v, \tau)$ where $u \in V(T_i)$ and $v \in V(T_j)$ with $i \leq j$, and $\tau \subseteq T \setminus \{T\}$.

**Input:** Vertices $u$ and $v$ where $u \in V(T_i)$ and $v \in V(T_j)$ for some $i \leq j$, and a set $\tau \subseteq T \setminus \{T\}$.

**Output:** A partial solution $F(u, v, \tau)$ for $(u, v, \tau)$ of minimum weight, or $\emptyset$ if not existent.

1: Let $S = \emptyset$.
2: for all $h \in [2]$ do
3:    if $A_h \cup T[x_1, u]$ is a path then
4:        if $v$ is reachable from $a_{3-h}$ in $G(A_h \cup T[x_1, u], a_{3-h}, v, \tau)$ then
5:            Compute a shortest $(a_{3-h}, v)$-path $R$ in $G(A_h \cup T[x_1, u], u, a_{3-h}, v, \tau)$.
6:        if $(A_h \cup T[x_1, u], R)$ is a partial solution for $(u, v, \tau)$ then
7:            $S \leftarrow (A_h \cup T[x_1, u], R)$.
8:    for all $i' \in [i-1]$ and $u' \in V(T_{i'})$ do
9:        for all $j' \in [i] \setminus [i']$ and $v' \in V(T_{j'})$ such that $T[x_{j'}, v'] \cap T[x_i, u] = \emptyset$ do
10:            for all $\tau' \subseteq \tau$ do
11:                if $F(u', v', \tau') = \emptyset$ then continue;
12:                Let $(Q_1', Q_2') = F(u', v', \tau')$.
13:                if $v$ is not reachable from $u'$ in $G(T[v', u], u', v, \tau \setminus \tau')$ then continue;
14:                Compute a shortest $(u', v)$-path $R$ in $G(T[v', u], u', v, \tau \setminus \tau')$.
15:                Let $P_1 = Q_2' \cup T[v', u]$ and $P_2 = Q_1' \cup R$.
16:                if $(P_1, P_2)$ is a partial solution for $(u, v, \tau)$ then
17:                    $S \leftarrow (P_1, P_2)$.
18:            if $S = \emptyset$ then return $\emptyset$.
19:        else let $S^*$ be the cheapest pair among those in $S$, and return $F(u, v, \tau) := S^*$.
The following lemma guarantees the correctness of Algorithm $\text{PartSol}$. Formally, we say that $F(u,v,\tau)$ is correctly computed if either it contains a minimum-weight partial solution for $(u,v,\tau)$, or no partial solution for $(u,v,\tau)$ exists and $F(u,v,\tau) = \emptyset$.

\begin{lemma}
Let $i,j \in [r]$ with $i \leq j$, $u \in V(T_i)$, $v \in V(T_j)$ and $\tau \subseteq T \setminus \{T\}$. Assuming that the values $F(u',v',\tau')$ are correctly computed for each $u' \in V(T_i')$ with $i' < i$, Algorithm $\text{PartSol}$ correctly computes $F(u,v,\tau)$.
\end{lemma}

Using the correctness of Algorithm $\text{PartSol}$, as established by Lemma 32, and the observation in Lemma 28 on how partial solutions can be used to find two permissively disjoint $(\{a_1,a_2\},\{b_1,b_2\})$-paths of minimum total weight, we obtain the following.

\begin{corollary}
For each constant $c \in \mathbb{N}$, there is a polynomial-time algorithm that finds two permissively disjoint $(\{a_1,a_2\},\{b_1,b_2\})$-paths of minimum total weight in $G$ (if such paths exist), where the set of negative edges in $G$ spans $c$ trees.
\end{corollary}

\section{Conclusion}

We have presented a polynomial-time algorithm for solving the Shortest Two Disjoint Paths problem on undirected graphs $G$ with conservative edge weights, assuming that the number of connected components in the subgraph $G[E^-]$ spanned by all negative-weight edges is a fixed constant $c$. The running time of our algorithm is $O(n^{2c+9})$ on an $n$-vertex graph. Is it possible to give a substantially faster algorithm for this problem? In particular, is it possible to give a fixed-parameter tractable algorithm for Shortest Two Disjoint Paths on undirected conservative graphs when parameterized by $c$?

More generally, is it possible to find in polynomial time $k$ openly disjoint $(s,t)$-paths with minimum total weight for some fixed $k \geq 3$ in undirected conservative graphs with constant values of $c$?

\section{References}


