Computers & Operations Research 153 (2023) 106151

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

A vertex-separator-based integer linear programming formulation for the partitioned Steiner tree problem

Mengfan Ma [a], Ziyang Men [b], André Rossi [c], Yi Zhou [a,*], Mingyu Xiao [a]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[b] Department of Computer Science and Engineering, University of California, Riverside, CA 92521, United States
[c] Université Paris-Dauphine, PSL, LAMSADE UMR 7243 CNRS, F-75016 Paris, France

ARTICLE INFO

Keywords:
Partitioned Steiner tree problem
Integer linear program
Vertex-separator
Branch-and-cut
Local branching

ABSTRACT

Given an undirected graph G with a cost function on vertices, a collection of subgraphs of G such that in each subgraph, there are some distinguished vertices called terminals, the Partitioned Steiner Tree Problem (PSTP) asks for a minimum cost vertex set such that, in each of the given subgraph G_i, the graph induced by the vertex set spans the terminal set in G_i. The PSTP generalizes the well-known Steiner tree problem and has important applications in computational sustainability, network design, and social network analysis. However, for solving the PSTP, conventional integer programming approaches based on single-commodity flow, multi-commodity flow and subtour elimination integer linear programs, suffer from low computational efficiency due to a substantial number of variables. In this paper, we propose a compact vertex-separator-based integer linear programming formulation with much fewer variables. Enhancing inequalities are also studied for tightening the formulation. We further investigate a branch-and-cut algorithm, a local-branching heuristic algorithm, and a hybrid algorithm combining them. In experiments where both public real-world and synthetic graphs are used, our hybrid algorithm outperforms all conventional approaches, especially for large graphs with more than ten thousand vertices. Further tests also validate the effectiveness of the proposed formulation and enhancing inequalities.

1. Introduction

Steiner problems, which require an optimal interconnection of a given set of objects with a predefined objective, are well-known in the operations research community. For example, the classic *Steiner Tree Problem* (STP) in graphs (Dreyfus and Wagner, 1971), which asks for a minimum cost tree spanning given terminals from a graph with edge costs, has been studied for decades (Klein and Ravi, 1995; Gouveia and Telhada, 2008; Buchanan et al., 2018). Applications of Steiner problems can be widely found in areas like network design (Held et al., 2011; Fischetti et al., 2017), databases (Ding et al., 2007) and social networks (Lappas et al., 2009). For readers who are interested in the recent developments on the STP, we refer to the comprehensive review in Ljubić (2021).

In this paper, we address a specific Steiner problem that arises in the field of computational sustainability in recent years (Lai et al., 2011; Brás et al., 2013; Brás and Cerdeira, 2015), namely, the *Partitioned Steiner Tree Problem* (PSTP). The input of the PSTP consists of an undirected graph $G = (V, E)$ with vertex set V and edge set E, a vertex cost vector $\boldsymbol{c} \in \mathbb{R}_+^{|V|}$ where c_v is the cost of $v \in V$, a collection of vertex subsets $\mathcal{V} = \{V_1, \ldots, V_p\}$ where p is the number of subsets, and a collection of terminal sets $\mathcal{T} = \{T_1, \ldots, T_p\}$ such that $T_k \subseteq V_k \subseteq V$ for all $k \in \{1, \ldots, p\}$. The objective of the PSTP is to find a subset of vertices $W \subseteq V$ such that for each $k \in \{1, \ldots, p\}$, and for each pair of terminals t and t' in T_k, there is a path between t and t' in the subgraph induced by $W \cap V_k$, i.e. $G[W \cap V_k]$, while the total cost of W, i.e. $\sum_{v \in W} c_v$, is minimized. Clearly, if any two subsets in $\mathcal{V}$ are disjoint, then the PSTP is equivalent to solving p independent instances of the vertex-cost STP. However, if one vertex subset in $\mathcal{V}$ intersects with another, we cannot solve the given problem by simply solving these p STP instances independently. An example of such a case is shown in Fig. 1, where two overlapping vertex subsets exist.

As far as we know, the PSTP stems from the real-world *corridor design problem* (Dilkina and Gomes, 2009) in the burgeoning computational sustainability area. The general purpose of corridor design is to establish wildlife corridors to connect fragmented habitats of different endangered species (Lai et al., 2011; Brás et al., 2013; Brás and Cerdeira, 2015). Specifically, a *land parcel* is a habitat where one or multiple species can reside and is represented by a vertex of the

* Corresponding author.
E-mail address: zhou.yi@uestc.edu.cn (Y. Zhou).

https://doi.org/10.1016/j.cor.2023.106151
Received 30 March 2022; Received in revised form 14 November 2022; Accepted 10 January 2023
Available online 24 January 2023

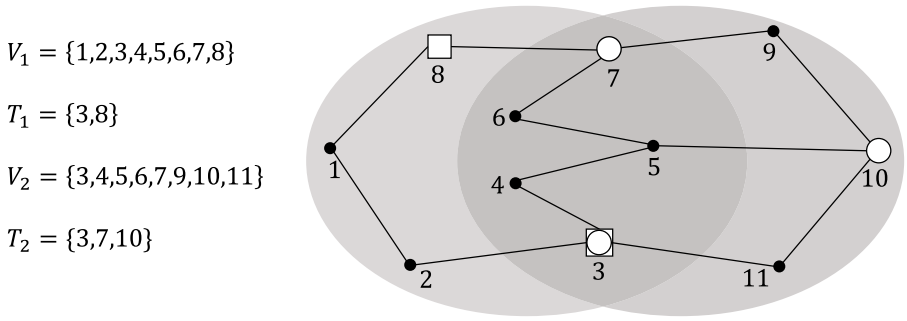


Fig. 1. In this example, all vertices have unit cost. Terminals in $T_1 \setminus T_2$ and $T_2 \setminus T_1$ are represented by white squares and white circles, respectively. Non-terminal vertices are those small black dots. Vertex 3, denoted by a white circle inside a white square, is the terminal in $T_1 \cap T_2$. The unique optimal solution for this PSTP instance is $\{4,5,6\} \cup T_1 \cup T_2$. For each $i \in \{1,2\}$, if we independently solve the vertex-cost STP instance with respect to induced graph $G[V_i]$ and terminal set T_i, the unique optimal solutions are $\{1,2\} \cup T_1$ and $\{9,11\} \cup T_2$, respectively.

input graph. If two land parcels share a border, then there is an edge connecting the corresponding vertices. The cost of protecting a land parcel such that animals can pass through it is indicated by its corresponding vertex cost. Different species may have different preferred land parcels due to the individual characteristics of the species. Also, two species may have overlapped preferred land parcels. Therefore, the terminal sets can be non-disjoint in the PSTP. Because the preferred land parcels may be fragmented due to human activities, we hope to (re)establish connectivity among land parcels for each endangered species. For economic purposes, we also hope to achieve this at the minimum cost.

Aside from computational sustainability, the PSTP also finds its applications in network protection, to protect communication network from geographical failures (Zhang et al., 2017; Pašić et al., 2021). Other applications exist in wireless sensor network design (Salhieh et al., 2001; Ali et al., 2016) and social network analysis (Faloutsos et al., 2004; Ahn et al., 2010; Fu et al., 2020).

In terms of complexity, the PSTP is NP-hard because it generalizes the well-known vertex-cost Steiner tree problem. In Lai et al. (2011), the PSTP is shown to be NP-hard even for planar graphs with only two terminal sets of size two. For practical purposes, some simple heuristics have been developed for solving the PSTP (Alagador et al., 2012; Brás et al., 2013; Brás and Cerdeira, 2015). These algorithms are fast but give no guarantee of optimality. As for exact algorithms that ensure optimality, to the best of our knowledge, there are only two exact approaches in the literature. These two approaches either model the PSTP into a Single-Commodity Flow (SCF) Integer Linear Program (ILP) or a Multi-Commodity Flow (MCF) ILP, and then solve the corresponding ILP by an integer programming solver (see Lai et al., 2011 for details of the encoding). Empirical evaluation of these ILP-based approaches disclosed that SCF and MCF can hardly solve real-world geometric graphs (typically, 2D grid graphs) with only a few hundred vertices in one hour, even when a state-of-the-art integer programming solver is used. Both formulations' performance is even unknown for social and communication network graphs. As a matter of fact, the existing formulations for the PSTP suffer from a vast number of variables, mainly due to the use of flow and edge cut constraints in these formulations. Hence, it is believed that the study of efficient algorithms to solve the PSTP is still in a rudimentary stage.

In this work, we focus on better ILP formulations and practical algorithms for the PSTP. Our contributions can be summarized as follows. We propose a more compact vertex-separator-based ILP formulation for the PSTP, in which only vertex variables are used. Connectivity is enforced by vertex-separator-based valid inequalities that can easily be separated. We optimally solve the ILP using a branch-and-cut algorithm. On top of this exact algorithm, we develop a local-branching-based heuristic. In addition, we also devise a hybrid algorithm that exploits the feasible solutions and useful cuts generated by the local-branching heuristic and solves the problem in the branch-and-cut framework. We conduct extensive experiments on public datasets and synthetic graphs, including general graphs instead of only grid graphs. The comparative results show that our vertex-separator-based algorithms outperform existing methods in the literature and can solve large instances of more than ten thousand vertices in less than one hour. In particular, to address instances of grid graph, we propose three new vertex-separators to enrich the polyhedral description of the problem, which may be relevant in a broader set of problems involving grids and connectivity constraints.

The remainder of the paper is organized as follows. Section 2 shows necessary notations and some preliminary observations. In Section 3, we present the vertex-separator-based ILP and two types of enhancing inequalities, the *neighbor inequalities* and the *2D grid inequalities*. Then, we give an exact branch-and-cut algorithm, a local-branching heuristic algorithm, and a hybrid algorithm, all based on our novel ILP, in Section 4. The computational results are presented in Section 5, including the comparative test and the parameter tuning procedures. In the last section, we draw the concluding remarks and highlight some directions for future works.

2. Notations and backgrounds

2.1. Notations

The input instance of the PSTP is denoted by $I = (G, c, p, \mathcal{V}, \mathcal{T})$.

- $G = (V, E)$ is an undirected graph with vertex set V and edge set E. For any $S \subseteq V$, denote by $G[S]$ the *subgraph of G induced by S*. The vertices in V are labeled as $\{1, 2, \ldots, |V|\}$.
- $c \in \mathbb{R}_+^{|V|}$ is a vertex cost vector where c_v is the cost of $v \in V$.
- p is a positive integer. We use $[p]$ to denote the *index set* $\{1, 2, \ldots, p\}$.
- $\mathcal{V} = \{V_1, \ldots, V_p\}$ is a collection of p vertex sets. For each $k \in [p]$, $V_k \subseteq V$ and $\bigcup_{k\in[p]} V_k = V$. Thus, $\mathcal{V}$ partitions G into p (possibly overlapping) subgraphs $G[V_1], \ldots, G[V_p]$. V_k is also called the kth *partition*. For convenience, we also use G_k to refer to $G[V_k]$. We denote by E_k the edge set of G_k. For any vertex $u \in V_k$, we denote by $N_k(u)$ the (open) neighborhood of u in G_k, that is, $N_k(u) = \{v \in V_k | (u, v) \in E_k\}$. The *closed neighborhood of u* in G_k is defined by $N_k[u] = N_k(u) \cup \{u\}$. Let $C \subseteq V_k$ be a vertex subset of V_k, we define the (open) neighborhood of C as $N_k(C) = \bigcup_{u\in C}(N_k(u) \setminus C)$.
- $\mathcal{T} = \{T_1, \ldots, T_p\}$ is a collection of p terminal sets where $T_k \subseteq V_k$ and $|T_k| \geq 2$ for $k \in [p]$. Let $T = \bigcup_{k\in[p]} T_k$.

The objective of the PSTP is to find a subset of vertices $W \subseteq V$ such that for each $k \in [p]$ and each pair of terminals t and t' in T_k, there exists a path between t and t' in the induced subgraph $G[W \cap V_k]$, while the total cost of W, i.e. $\sum_{v\in W} c_v$, is minimized.

By a simple preprocessing procedure, we can merge adjacent terminals of the same partition. Thus we can assume without loss of generality that no two terminals in the same partition are adjacent. Then we have the following simple observations.

Observation 1. *If $V_k = V$ for all $k \in [p]$, then the PSTP is equivalent to the classical Steiner Tree Problem on V, with $T = \bigcup_{k=1}^{p} T_k$.*

Observation 2. *If $V_p \subseteq V_{p-1}$ and $T_p \subseteq T_{p-1}$, then any solution spanning T_{p-1} also spans T_p, thus partition V_p and T_p can be removed from the problem input without impacting optimal solutions.*

Observation 3. *If $\bigcup_{k=1}^{p} V_k = V$, $V_k \cap V_{k'} = \emptyset$ for all $k, k' \in [p]$ with $k \neq k'$, then the PSTP is equivalent to p independent vertex-cost Steiner Tree Problems.*

Observation 4. *For a certain integer $p' \in [p-1]$, if $\bigcup_{k=1}^{p'} V_k$ does not intersect with $\bigcup_{p'+1}^{p} V_k$, then the original PSTP is equivalent to 2 independent PSTPs.*

2.2. Vertex separators

In this subsection, we introduce the notions of *vertex-separator* and *terminal-separator* which are basics of our ILP formulation.

- **connected component.** A *connected component* (see in West et al., 2001) of an undirected graph $G = (V, E)$ is a vertex set $C \subseteq V$ such that the induced graph $G[C]$ is connected and $G[C \cup \{v\}]$ is not connected for any $v \in V \setminus C$.
- (u, v)**-separator**. For two distinct u and v in V_k, the set $S \subseteq V_k \setminus \{u, v\}$ is called a (u, v)*-separator in* G_k if the removal of S from G_k separates u and v into distinct connected components. A (u, v)-separator S is *minimal* if any proper subset of S is no longer a (u, v)-separator. For any $u, v \in V_k$, let $\mathcal{S}_{uv}^k$ be the family of all (u, v)-separators in G_k.
- **terminal-separator.** We call a (u, v)-separator in G_k a *terminal-separator* if it separates two distinct terminals $t, t' \in T_k$.

3. Integer linear programming formulations of the PSTP

In this section, we first introduce our vertex-separator-based ILP formulation, the VS. Then we briefly compare the VS formulation and the other three existing ILP formulations in terms of their variables and constraints numbers. Finally, we study more inequalities for tightening the VS model in the branch-and-cut method.

3.1. The vertex-separator-based ILP

In recent years, compact vertex-separator-based ILP formulations have gained popularity in modeling some connectivity problems. For example, this type of model plays an essential role in the state-of-the-art methods of solving the Steiner tree problem in Fischetti et al. (2017). Also, in Bley et al. (2017), the vertex-separator model is used to solve the node-weighted dominating Steiner problem. Our ILP model for the PSTP generally follows this scheme. Let us show the VS model in the followings.

$$\text{minimize} \sum_{u \in V} c_u x_u \tag{1}$$

$$x_t = 1 \qquad \forall t \in T \tag{2}$$

$$\sum_{u \in S} x_u \geq 1 \qquad \forall k \in [p], \forall t, t' \in T_k \text{ and } t < t', \forall S \in \mathcal{S}_{tt'}^k \tag{3}$$

$$x_u \in \{0, 1\} \qquad \forall u \in V \tag{4}$$

In the VS, each vertex $u \in V$ is associated with a binary variable x_u such that x_u is set to 1 if u is part of the solution, and 0 otherwise. Constraint (2) enforces that each terminal must be selected in the solution. Constraint (3) is the *connectivity constraint* which requires that terminals in each G_k must be pairwise connected. The connectivity constraint is based on the important property that, to connect two terminals t and t' in T_k, at least one vertex from every minimal (t, t')-separator in G_k must be selected in the solution.

Table 1
The sizes of different ILPs.

Formulation	No. Var.	No. Cons.
Single-Commodity Flow (Lai et al., 2011)	$O(p(\|V\| + 2\|E\|))$	$O(p(\|V\| + 2\|E\|))$
Multi-Commodity Flow (Lai et al., 2011)	$O(p\|T\|(\|V\| + 2\|E\|))$	$O(p\|T\|\|V\|)$
Steiner Tree (Dilkina and Gomes, 2010)	$O(2p\|E\| + \|V\|)$	$O(p\|V\|2^{\|V\|})$
Vertex-Separator-Based Model (This paper)	$O(\|V\|)$	$O(p\|T\|^2 2^{\|V\|})$

3.2. Comparing the VS with existing ILPs

Existing ILPs for the PSTP include Single-Commodity Flow (Lai et al., 2011), Multi-Commodity Flow (Lai et al., 2011) and Steiner Tree (Dilkina and Gomes, 2010) formulations. For completeness, we leave the detailed description of these three formulations to Appendix B in Appendix A. It is worth noticing that, in order to build up these formulations, the input undirected graph is first converted into a directed graph by replacing each edge with two reverse directed arcs. Then each directed arc is associated with a binary variable. As a consequence, the number of variables of these formulations is proportional to $|E|$. However, $|E| \gg |V|$ in many graphs. Thus, it is natural to think about more compact formulations in which each variable is associated with a vertex. It turns out that our Vertex-Separator (VS) based formulation meets such an expectation. In Table 1, we summarize the number of variables and constraints for all these formulations. It can be seen that the VS formulation has much fewer variables than existing ones. Note that, though the VS formulation has an exponential number of constraints, they can be separated by fast polynomial algorithms, as will be seen later.

3.3. Enhancing the VS formulation

Due to the exponential number of inequalities in the connectivity constraint, the VS model is solved by a branch-and-cut approach. The root node of the branch-and-cut only contains a partial VS formulation without the connectivity constraint, *i.e.*, objective function (1), constraints (2) and (4). To tighten the initial VS formulation, we can supplement it with additional inequalities. This section is dedicated to these additional inequalities.

3.3.1. Neighbor inequalities

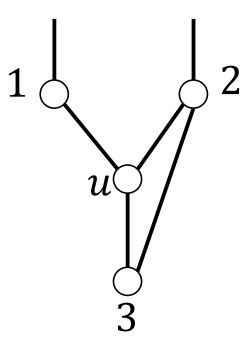
We first introduce the *neighbor inequalities* to enhance the partial VS model on the root node. Given a subgraph G_k, for any terminal $t \in T_k$, it is obvious that t's neighborhood $N_k(t)$ is a terminal-separator. Constraint (5) ensures that at least one of t's neighbors is selected in the solution. Obviously, constraint (5) is just a special case of the connectivity constraint (3), but it can be used in the branch-and-cut as a part of the initial constraints.

$$\sum_{v \in N_k(t)} x_v \geq 1 \qquad \forall k \in [p], \forall t \in T_k \tag{5}$$

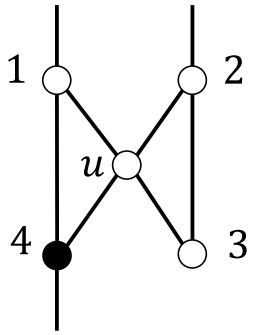
For a non-terminal vertex $u \in V_k$, if u is selected, then at least two vertices in $N_k(u)$ have to be selected since otherwise we can delete u and obtain an improved solution due to the fact that $c_u \geq 0$. Therefore, we obtain constraint (6) for each non-terminal vertex in V_k. Constraint (6) is not contained in the connectivity constraint (3) and can strengthen the VS model.

$$\sum_{v \in N_k(u)} x_v \geq 2x_u \qquad \forall k \in [p], \forall u \in V_k \setminus T_k \tag{6}$$

Constraint (6) can be further tightened. For a non-terminal vertex $u \in V_k$, let us consider the set of non-terminals in $N_k(u)$ which are only adjacent to $N_k[u]$, *i.e.*, $S_k = \{v \in N_k(u) \setminus T_k \mid N(v) \subseteq N[u]\}$. It can



(a) (6): $x_1 + x_2 + x_3 \geq 2x_u$
(7): $x_1 + x_2 \geq 2x_u$



(b) (6): $x_1 + x_2 + x_3 + x_4 \geq 2x_u$
(8): $x_2 \geq x_u$

Fig. 2. Strengthened neighbor inequalities for non-terminal u. The white vertices represent the non-terminals, and the black vertices are the terminals.

be observed that if u is selected, then no vertex of S_k is selected for spanning T_k in an optimal solution. Therefore, we can replace $N_k(u)$ by $N_k^*(u) = N_k(u) \setminus S_k$ in constraint (6) and obtain constraint (7), as illustrated in Fig. 2(a).

$$\sum_{v \in N_k^*(u)} x_v \geq 2x_u \qquad \forall k \in [p], \forall u \in V_k \setminus T_k \tag{7}$$

We can observe that constraints (5) and (6) are similar to the node-degree inequalities in Fischetti et al. (2017), and constraint (7) is similar to strengthened node-degree inequality in Fischetti et al. (2017). Next we show that constraint (7) can be further strengthened by considering the particular case where $u \in V_k \setminus T_k$ is adjacent to a terminal $t \in T_k$. If u is in the solution, then at least one vertex in $N_k^*(u)$ should also be selected in order to connect u to another vertex out of the neighborhood of t. This property is enforced by constraint (8), and is illustrated in Fig. 2(b).

$$\sum_{v \in N_k^*(u) \setminus N_k[t]} x_v \geq x_u \qquad \forall k \in [p], \forall t \in T_k, \forall u \in N_k(t) \tag{8}$$

Constraint (8), which does not appear in Fischetti et al. (2017) is now shown to dominate constraint (7) for all $t \in T_k$, and for all $u \in N_k(t)$. First, we observe that t does not belong to S_k, hence t is in $N_k^*(u)$. Consequently, $(N_k^*(u) \setminus N_k[t]) \cup \{t\} = N_k^*(u) \setminus N_k(t)$. We also have $x_t \geq x_u$, because $x_t = 1$ for all $t \in T_k$. Adding this inequality to constraint (8) yields:

$$\sum_{v \in N_k^*(u) \setminus N_k(t)} x_v \geq 2x_u \qquad \forall k \in [p], \forall t \in T_k, \forall u \in N_k(t)$$

Since $N_k^*(u) \setminus N_k(t) \subseteq N_k^*(u)$, constraint (8) implies constraint (7). We now use the example of Fig. 2(b) to show that these constraints can be different. Indeed constraint (7) is $x_1 + x_2 + x_4 \geq 2x_u$, whereas constraint (8) is $x_1 \geq x_u$.

If u is not adjacent to any terminal, we rewrite constraint (7) as constraint (9) for completeness.

$$\sum_{v \in N_k^*(u)} x_v \geq 2x_u \qquad \forall k \in [p], \forall u \in V_k \setminus \bigcup_{t \in T_k} N_k[t] \tag{9}$$

In our implementation, we enforce constraints (5), (8) and (9) in the partial VS model at the root node.

3.3.2. 2D grid inequalities

Since the PSTP is particularly useful in wildlife conservation where the given graphs are often *2-dimensional (2D) grid graphs*, also known as mesh graphs (Lai et al., 2011; Dilkina et al., 2013), we investigate inequalities which are specifically helpful for such graphs.

First, it is known that a 2D grid graph $G = (V, E)$ can be represented in the plane, where each vertex is indexed by two-dimensional coordinates. Fig. 3 shows an example of three 8×8 2D grid graphs. Assuming that the graph fits into a $n \times n$ square, we use $\langle i, j \rangle$ $(i, j \in [n])$ to index the vertex located at the ith row and jth column.

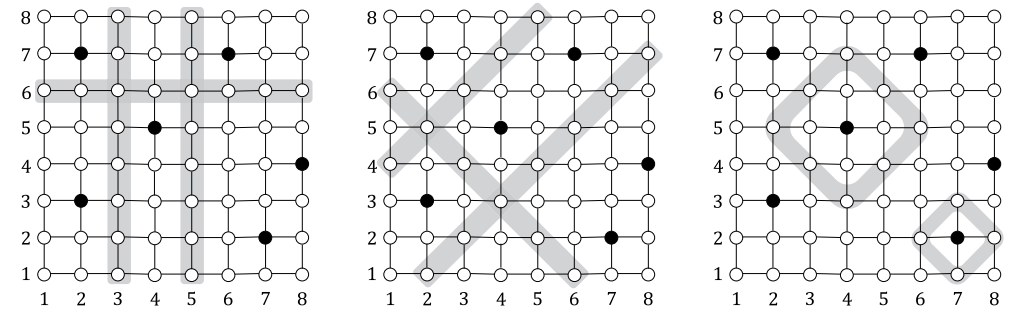
Given a PSTP instance, let us assume that for each $k \in [p]$, $G_k = (V_k, E_k)$ is a 2D grid graph, then we can define the following new separators for G_k.

- **Row and column separators** A *row* is a set of vertices that share the same first coordinate. In G_k, a *row separator* R is a row and also a terminal-separator that contains no vertex in T_k. Likewise, a *column* is a set of vertices which have the same second coordinate and a *column separator* C is a column and a terminal-separator that contains no vertex in T_k. According to the given definitions, the first and last rows are not row separators. Likewise, the first and last columns are not column separators. Fig. 3(a) is an example of G_k where black vertices represent terminals, and the gray strips highlight row and column separators.
- **Diagonal and anti-diagonal separators** A *diagonal* is a set of diagonal vertices in G_k. Formally, $D = \{\langle i, 1\rangle, \langle i+1, 2\rangle, \ldots, \langle n, n-i+1\rangle\}$ for any $i \in [n]$ or $D = \{\langle 1, j\rangle, \langle 2, j+1\rangle, \ldots, \langle n-j+1, n\rangle\}$ for any $j \in [n]$ is a diagonal. A *diagonal separator* D is a diagonal and also a terminal-separator that contains no vertex in T_k. An *anti-diagonal* is a set of anti-diagonal vertices in G_k, defined as $A = \{\langle i, j\rangle \in V_k | i, j \in [p] \text{ and } i + j = s\}$ for any $s \in \{2, \ldots, 2n\}$. Again, we define the *anti-diagonal separator* as an anti-diagonal which is a terminal-separator and contains no vertex in T_k. In Fig. 3(b), the gray strips show diagonal and anti-diagonal separators.
- **d-neighborhood separator** Let $\langle i, j\rangle$ and $\langle i', j'\rangle$ be the indices of two vertices in a 2D grid graph, we define the *shortest distance* between them as $|i - i'| + |j - j'|$. Given a positive integer d, the *d-neighborhood* of a terminal $t \in T_k$ is then the set of vertices in V_k whose shortest distance to t is d. A d-neighborhood of t is a *d-neighborhood separator* if it contains no terminal vertices and separates t and another terminal in T_k. In Fig. 3(c), the gray circles show a 1-neighborhood separator and a 2-neighborhood separator.

For a 2D graph, it is easy to list all the row, column, diagonal, anti-diagonal, 1-neighborhood, and 2-neighborhood separators. Given subgraph G_k and terminal set T_k in a PSTP instance, denote by S_{grid}^k the collection of all these separators. Then we have the following *grid inequalities*:

$$\sum_{u \in S} x_u \geq 1 \qquad \forall k \in [p], \forall S \in S_{grid}^k \tag{10}$$

It is worth mentioning that inequalities (10) cannot tighten the VS model. Row separators, column separators and diagonal separators are also vertex separators in 2D graphs. However, grid inequalities are



(a) one row and two column separators (b) two diagonal and one anti-diagonal separators (c) two neighborhood separators

Fig. 3. New separators for grid graph.

helpful to speed up the branch-and-cut with the VS model. Specifically, when the given graph is a 2D grid graph, we produce all grid inequalities and add them to the root node (where a partial VS model without connectivity constraints is solved). The number of row and column separators is at most $2p|V|$, the number of diagonal and anti-diagonal separators is at most $4p|V|$, and the number of 1-neighborhood and 2-neighborhood separators is at most $2p|T|$.

4. Algorithms for the PSTP

In this section, we introduce algorithms based on the VS formulation. As mentioned, a branch-and-cut algorithm is first designed to obtain an optimal solution of the PSTP. Then, we develop a local-branching-based heuristic algorithm, and a hybrid algorithm which combines local-branching heuristic and the branch-and-cut algorithm.

4.1. An exact branch-and-cut algorithm

The VS model includes an exponential number of connectivity inequalities, *i.e.*, inequalities in constraint (3). However, these connectivity inequalities can be separated in polynomial time. Therefore, we resort to the branch-and-cut framework (Wolsey, 2020) for addressing a relaxed model that is enriched by violated constraints on the fly.

Let us introduce the general procedure of the branch-and-cut framework for the VS model. In what follows, a *partial VS model* is made up of objective function (1), constraints (2), (4), and the neighbor inequalities constraints (5), (8) and (9), with only a few inequalities of constraint (3). Initially, the root node of the branch-and-cut contains a partial VS model that excludes other inequalities of constraint (3), that is, objective function (1) and constraints (2), (4), (5), (8) and (9). When a 2D grid graph is given, constraint (10) in Section 3.3.2 is also included. We call this model the *initial partial VS model*.

Then, the branch-and-cut algorithm solves the initial partial VS model at the root node and adds the violated connectivity inequalities and other cuts on the fly at each branch node. Specifically, when an integer or even a fractional solution is found for a partial VS model, the algorithm addresses the following problem using a *separation oracle*:

Given a current solution $\boldsymbol{x}$ of a partial VS model, either find out at least one minimal terminal-separator S in G_k for a certain $k \in [p]$ such that $\sum_{u\in S} x_u < 1$, or return "satisfied" if there is no such vertex separator for any $k \in [p]$.

Because a solution $\boldsymbol{x}$ of a partial VS model can be either integral or fractional, we introduce separation methods for integer solutions and fractional solutions in Section 4.1.1 and Section 4.1.2, respectively. Then in Section 4.1.3, we combine these two methods in our integrated separation oracles.

Algorithm 1: The separation method for integer solutions

Input: A restricted solution $\boldsymbol{x}_k$, graph G_k, parameters ϵ and α
Output: A queue of inequalities that violate constraint (3)

```
Let Q be the set of violated inequalities, Q ← ∅ initially.
Build the support graph G_k^x from x_k
Apply Depth-First Search (DFS, see Tarjan (1972)) to G_k^x to
  find out the connected component(s)
/* Since each connected component contains at
   least one terminal, we denote a connected
   component by C_t, where t is an arbitrary
   terminal in the connected component */
if there is more than one connected component then
    foreach distinct pair of connected components C_t and C_t'
      such that t < t' do
        Delete all edges in G[C_t ∪ N_k(C_t)] from G_k
        Find the set R_t' of vertices that are reachable from t'
          in G_k via DFS
        S ← N_k(C_t) ∩ R_t'
        Add Σ_{u∈S} x_u ≥ 1 to Q
        Recover all deleted edges in line 6
    end
end
Remove inequalities in Q with a violation less than ε
Q ← the first α most violated inequalities in Q
return Q
```

4.1.1. Separation by connected components

Assume that the current solution $\boldsymbol{x}$ is integral, that is, all the elements of $\boldsymbol{x}$ are integers. We use a similar separation procedure for integer solutions as in Fischetti et al. (2017) but run it separately for each subgraph. We define $\boldsymbol{x}_k$ as the *restricted solution* on G_k, that is, $\boldsymbol{x}_k$ is obtained from $\boldsymbol{x}$ by removing all x_u if $u \notin V_k$. For all $k \in [p]$, we build the *support graph* $G_k^{\boldsymbol{x}} = (V_k^{\boldsymbol{x}}, E_k^{\boldsymbol{x}})$ where $V_k^{\boldsymbol{x}} = \{u \in V_k | x_u = 1\}$ and $E_k^{\boldsymbol{x}} = \{(u, v) \in E_k | x_u = x_v = 1\}$. Then by Algorithm 1 (Tarjan, 1972), we use the restricted solution $\boldsymbol{x}_k$ to separate the connectivity inequalities. It has been proved in Fischetti et al. (2017) that S in line 8 of Algorithm 1 is a minimal (t, t')-separator.

Moreover, as mentioned in Taccari (2016), Algorithm 1 can be used to separate the connectivity inequalities at a fractional node of the branch-and-bound tree search: given a fractional restricted solution $\boldsymbol{x}_k$, for each fractional $x_u \in \boldsymbol{x}_k$, we set x_u to 1 if x_u is greater than 0.01 and 0 otherwise. Through this, we convert a fractional $\boldsymbol{x}_k$ to an integral one and then we can build a support graph from it. When Algorithm 1 terminates, we only return the first α most violated inequalities with a violation at least ϵ. We note that the separation procedure of Algorithm 1 is not guaranteed to find all the violated inequalities on fractional

solutions, but the correctness is preserved by the fact that the procedure is exact for integer solutions. Without much effort, one can check that time complexity of Algorithm 1 is bounded by $O(|V_k| + |E_k|)$.

4.1.2. Separation by minimum cuts

When the solution $\boldsymbol{x}$ is fractional, the connectivity constraint can also be separated by a network flow algorithm, as stated in Fischetti et al. (2017). We present such a separation algorithm for the PSTP in Algorithm 2. First a *support flow network*, *i.e.*, a digraph with edge capacity, is built based on $\boldsymbol{x}_k$. Then, we find the minimum cuts for all distinct pairs of terminals in the support flow network. The parameters ϵ and α are also used to filter the violated inequalities, as in Algorithm 1.

Algorithm 2: The separation method for fractional solutions

Input: A fractional $\boldsymbol{x}_k$, graph G_k, parameters ϵ and α
Output: A queue of inequalities that violate constraint (3)

Let Q be the set of violated inequalities, $Q \leftarrow \emptyset$ initially.
Replace each edge (u, v) in E_k by two arcs (u, v) and (v, u)
Replace each $u \in V_k$ by a *vertex-arc* (u_{in}, u_{out}), with capacity x_u
For each $u \in V_k$, all arcs entering u are now directed into u_{in}, and all arcs leaving u are now directed out of u_{out}, capacity of these arcs are set to ∞
/* Denote by $F_k^{\boldsymbol{x}}$ the constructed flow network. */
foreach *distinct terminal pair* (t, t') *in* T_k **do**
  Find a minimum (t_{out}, t'_{in})-cut C_{min} in $F_k^{\boldsymbol{x}}$
  Replace the vertex-arcs in C_{min} by their corresponding vertices
  if $\sum_{u \in C_{min}} x_u < (1 - \epsilon)$ **then**
    Add $\sum_{u \in C_{min}} x_u \geq 1$ to Q
  end
end
$Q \leftarrow$ the first α most violated inequalities in Q
return Q

We use the well-known Goldberg-Tarjan's highest-label preflow-push algorithm (Goldberg and Tarjan, 1988) to solve the minimum cut problem. Thus the running time of Algorithm 2 is $O(|T_k|^2|V_k|^2\sqrt{|E_k|})$. We note that given a fractional solution, this separation procedure practically finds more violated connectivity inequalities than Algorithm 1, but with a higher computational cost.

4.1.3. The integrated separation oracle

As mentioned before, Algorithm 1 can separate both integer and fractional solutions, while Algorithm 2 only separates fractional solutions. We observe that even in a fractional solution $\boldsymbol{x}$, there may exist an $\boldsymbol{x}_k$ containing only integers for a certain $k \in [p]$, which can be separated by the faster Algorithm 1. To fully exploit both algorithms and achieve the best performance over all different types of PSTP instances, we design our separation oracle by combining the two algorithms in different manners, as shown in Algorithm 3. Algorithms 1,2 are denoted by ConnectedSeparation and FlowSeparation, respectively. See Appendix A for a basic framework of our branch-and-cut method.

Algorithm 3: The integrated separation oracle

Input: A PSTP instance $I = (G, c, p, \mathcal{V}, \mathcal{T})$, a current solution $\boldsymbol{x}$, parameter sets (ϵ_1, α_1) and (ϵ_2, α_2)
Output: A queue of inequalities that violate constraint (3)

Let Q be the set of violated inequalities, $Q \leftarrow \emptyset$ initially.
if $\boldsymbol{x}$ *is integral* **then**
  foreach $k \in [p]$ **do** $Q \leftarrow Q \cup$ ConnectedSeparation$(\boldsymbol{x}_k, G_k, \epsilon_1, \alpha_1)$;
end
if $\boldsymbol{x}$ *is fractional* **then**
  foreach $k \in [p]$ **do**
    if $\boldsymbol{x}_k$ *is integral* **then**
      $Q \leftarrow Q \cup$ ConnectedSeparation$(\boldsymbol{x}_k, G_k, \epsilon_1, \alpha_1)$
    else
      if *ConnectedSeparation*$(\boldsymbol{x}_k, G_k, \epsilon_1, \alpha_1)$ *is not empty* **then**
        $Q \leftarrow Q \cup$ ConnectedSeparation$(\boldsymbol{x}_k, G_k, \epsilon_1, \alpha_1)$
      else
        $Q \leftarrow Q \cup$ FlowSeparation$(\boldsymbol{x}_k, G_k, \epsilon_2, \alpha_2)$
      end
    end
  end
end
return Q

Algorithm 4: The local branching algorithm for the PSTP

Input: A PSTP instance $I = (G, c, \mathcal{V}, \mathcal{T}, p)$, lower and upper bound for radius r_{min} and r_{max}, radius step r_δ, maximum number of iteration $LBMaxIter$, time limit $LBTimeLim$
Output: A high-quality solution and an enlarged $CutPool$

$itr \leftarrow 1$, $r \leftarrow r_{min}$, $CutPool \leftarrow \emptyset$
Build an integer solution $\boldsymbol{x}$ by the constructive heuristic
while $(itr \leq LBMaxIter)$ *and* $(r \leq r_{max})$ **do**
  Add all the inequalities of constraint (11) to the initial VS model
  $(\boldsymbol{x}', CutPool) \leftarrow$ B&C(I, $\boldsymbol{x}$, $CutPool$, $LBTimeLim$, $LBMaxIter$) /* All the newly generated connectivity inequalities are stored in $CutPool$ */
  if $\sum_{u \in V} c_u x'_u < \sum_{u \in V} c_u x_u$ **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
    $r = r_{min}$
  else
    $r \leftarrow r + r_\delta$
  end
  $itr \leftarrow itr + 1$
end
return $(\boldsymbol{x}, CutPool)$

4.2. A local-branching-based heuristic algorithm

The local-branching (LB) based algorithm, proposed by Fischetti and Lodi (2003), is a powerful heuristic search algorithm. LB follows the large-neighborhood search scheme and uses the ILP solver to drive the local search.

Our LB-based heuristic for the PSTP is detailed in Algorithm 4. First, an initial solution $\boldsymbol{x}$ is built by a fast constructive heuristic. Specifically, for each G_k, the constructive heuristic iteratively builds a shortest path between two randomly selected terminals until all the terminals are connected. Then, all the vertices on these shortest paths are selected in the solution. It is known that Lai et al. (2011) designed a more complex Iterative Dynamic Programming heuristic for the PSTP, which is based on the exact Dreyfus–Wagner Dynamic Programming algorithm (Dreyfus and Wagner, 1971). The running time of the Iterative Dynamic Programming is exponential in $|T|$. Thus we choose to use the simpler and faster constructive heuristic.

When an initial feasible solution is obtained, an iterative local search procedure is used to improve the quality of the solution. The local search is depicted in the *while* loop of Algorithm 4. Given a feasible solution $\boldsymbol{x}$, for each $k \in [p]$, let $W_k = \{u \in V_k | x_u = 1\}$ be the set of selected vertices in V_k. An *r-distance neighbor solution* of $\boldsymbol{x}$ is defined as a new feasible solution such that for each $k \in [p]$, the vertices in

V_k that are selected by the new solution differ from W_k by at most r vertices. In order to get an r-distance neighbor solution, in line 4 we add the following constraint (11) to the initial VS model.

$$\sum_{u \in W_k} (1 - x_u) \leq r \quad \forall k \in [p] \tag{11}$$

Besides constraint (11), Fischetti et al. introduced another notion of r-distance neighbor solution in Fischetti et al. (2017) and formalized constraint (12). For a fixed radius r, constraint (11) dominates constraint (12). After some preliminary tests, we use constraint (11) in our algorithm as it leads to better overall performance.

$$\sum_{u \in V_k \setminus W_k} x_u + \sum_{u \in W_k} (1 - x_u) \leq r \quad \forall k \in [p] \tag{12}$$

In line 5, we reuse the branch-and-cut algorithm (denoted by B&C) in Section 4.1 to search for an r-distance neighbor solution of smaller objective value. The initial solution $\boldsymbol{x}$ is given to the branch-and-cut as a warm-start solution. A cutting-off time of $LBTimeLim$ is also given in case the branch-and-cut search does not stop in a reasonable time. Besides, inequalities of the connectivity constraint found by the separation oracle are stored in a global $CutPool$. The inequalities in the $CutPool$ are reused in the subsequent branch-and-cut.

From line 6 to line 12, we update the current solution and the current radius r. The radius r, initialized to r_{min}, is increased by r_δ if no improved solution is found by the branch-and-cut in the current iteration. If a better neighbor solution $\boldsymbol{x}'$ is obtained, r is reset to r_{min} and the current solution is updated to $\boldsymbol{x}'$.

The algorithm stops when a given number of iterations, *LBMaxIter*, is reached, or r exceeds the maximum radius r_{max}. Note that the final $CutPool$ will be reused for our hybrid algorithm in the next subsection.

4.3. A hybrid algorithm

In this section, we introduce a hybrid algorithm that combines our branch-and-cut and LB-based heuristic. The overall algorithmic framework of our hybrid algorithm is detailed in Algorithm 5. In general, the hybrid algorithm consists of two phases. In the first phase, the LB-based heuristic algorithm is called multiple times to obtain a high-quality solution. Meanwhile, the LB-based heuristic also collects a set of valid connectivity inequalities and keeps them in the $CutPool$. In the second phase, the branch-and-cut is started with the initial partial VS model enriched with the inequalities in the $CutPool$ that is obtained in the first phase, with the purpose to obtain an optimal solution.

In Algorithm 5, we use $\boldsymbol{x}$ to store the best solution found so far. From line 2 to 8, the LB-based heuristic (denoted by LB) in Section 4.2 is called repeatedly until the time or the number of iterations exceeds the limit. The LB-based heuristic produces not only a high-quality solution $\boldsymbol{x}$, but also a set of valid inequalities in $CutPool$. Because these inequalities are globally valid, they are added to the VS model iteratively each time the LB-based heuristic is called (line 3). If the LB-based heuristic finds an improved solution, we update the best solution and proceed to the next iterative call of the LB-based heuristic (line 4 to 7). After the iterative calls of the local branching heuristic, we obtain a solution $\boldsymbol{x}$ and a set of connectivity inequalities in *CutPool*. They are again given to the branch-and-cut algorithm as a warm-start and a set of additional inequalities to be added to the initial partial VS model. Then in line 9, the branch-and-cut algorithm is executed until an optimal solution is reached or $LBTimLim$ seconds have elapsed.

Algorithm 5: The hybrid algorithm

Data: A PSTP instance $I = (G, c, \mathcal{V}, \mathcal{T}, p)$, restart limit $LBMaxRestarts$, time limit $TimeLim$, local branching parameters (r_{min}, r_{max}, r_δ, $LBMaxIter$, $LBTimeLim$)
Result: A (sub)-optimal solution $\boldsymbol{x}$.

1 $itr \leftarrow 1, \boldsymbol{x} \leftarrow (1, \ldots, 1), CutPool \leftarrow \emptyset$
2 **while** $itr \leq LBMaxRestarts$ *and the running time is within* $TimeLim$ **do**
3 $(\boldsymbol{x}', CutPool) \leftarrow$ LB(I, $CutPool$, r_{min}, r_{max}, r_δ, $LBMaxIter$)
4 **if** $\sum_{u \in V} c_u x'_u < \sum_{u \in V} c_u x_u$ **then**
5 $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
6 **end**
7 $itr \leftarrow itr + 1$
8 **end**
9 $\boldsymbol{x} \leftarrow$ B&C(I,$\boldsymbol{x}$, $CutPool$, $LBTimeLim$)
10 **return** $\boldsymbol{x}$

5. Experiments

5.1. Settings

All the algorithms are implemented in C++ on a PC with an Intel i5 processor at 3.20 GHz and 16 GB RAM[1]. The operating system is Microsoft Windows 10, 64 bits. We use IBM ILOG CPLEX 12.71 as our underlying ILP solver. The settings by default of CPLEX are used unless otherwise specified.

We use the Multi-Commodity Flow (MCF) formulation (Lai et al., 2011) and the Steiner tree formulation (See Appendix B.3 for details of both formulations) as existing benchmark algorithms. Precisely speaking, the tested algorithm is the CPLEX solver that uses a specific formulation. Note that the Single-Commodity Flow (SCF) formulation is not compared because it is dominated by the MCF, as shown in Lai et al. (2011). For the MCF, we use the entire model to solve the PSTP since the MCF is of polynomial-size and in Lai et al. (2011) the entire model is also used. For the Steiner tree formulation, considering that it contains exponentially many inequalities, we also use a branch-and-cut algorithm to solve it, with the min-cut separation approach stated in Dilkina and Gomes (2010) and Taccari (2016).

5.1.1. Parameter settings

In the experiments, we set $\epsilon = 0.001$ and α to ∞ (namely, we add all the inequalities with a violation at least 0.001) for Algorithm 1. As for Algorithm 2, we set $\epsilon = 0.4$ and $\alpha = 20$. We also set $r_{min} = 10$, $r_{max} = 30$, $r_\delta = 2$, $LBMaxIter = 10$ and $LBTimeLim = 10$ seconds. The parameter tuning procedure is summarized in Appendices C.1 and C.2 in Appendix A.

5.1.2. Benchmark instances

We use five benchmark sets from different sources.

- **SNAP Instances.** We collect 12 graphs from the Stanford Network Analysis Platform(SNAP)[2]. For each graph, the required inputs of a PSTP instance $I = (G, c, \mathcal{V}, \mathcal{T}, p)$ are built as follows. Parameter p is $\frac{|V|}{1000}$. Each $|V_k|$ is generated randomly according to a uniform distribution over the closed interval $[0.3|V|, 0.7|V|]$ and $|T_k|$ is $\lceil \frac{|V_k|}{100} \rceil$. To obtain each partition set V_k, we collect all the visited vertices in a Breath-First Search on G, starting from a randomly chosen vertex in V until the number of required vertices in that partition is reached. Terminal set T_k is then generated randomly according to a uniform distribution over V_k.
- **SteinLib Instances.** In total, 53 graphs are collected from the SteinLib Testdata Library (SteinLib)[3]. These are all small-sized STP instances, from sparse to very dense. An original graph $G = (V, E, T)$ is turned into a PSTP instance as follows: p is generated randomly according to a uniform distribution over $\{2, 3, 4\}$, each $|V_k|$ is generated randomly according to a uniform distribution

[1] The code is available at https://github.com/Mark-htmlgogogo/Steiner-Multigraph-Problem

[2] https://snap.stanford.edu/data/

[3] http://steinlib.zib.de/steinlib.php

Table 2
Results for instances from SNAP.

Ins.(12)	$\|V\|$	$\|E\|$	$\|P\|$	$\|V_k\|$	$\|T_k\|$	MCF(6)			Steiner(5)			VS-BC(9)			VS-LB(9)			VS-Hybrid(12)	
						Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj
email-Eu-core	986	16064	3	493	5	12.5	54573*	0.00	27.3	54573*	0.00	1.2	54573*	0.00	**0.6**	**54573***	**0.00**	0.9	54573*
CollegeMsg	1899	13838	2	950	9	232.4	87859*	0.00	432.4	87859*	0.00	1.8	87859*	0.00	**1.1**	**87859***	**0.00**	0.9	87859*
soc-sign-bitcoinalpha	3783	14124	4	1890	19	OT	179063	7.46	OT	264454	58.71	OT	166750	0.07	**0.4**	**166629***	**0.00**	1.0	166629*
ego-Facebook	4039	88234	4	2020	20	**1.8**	**378283***	**0.00**	186.4	378283*	0.00	23.6	378283*	0.00	16.3	378283*	0.00	19.9	378283*
ca-GrQc	5241	14484	5	2615	26	OT	N/A	INF	OT	N/A	INF	73.6	234121*	0.00	**13.8**	**234121***	**0.00**	15.8	234121*
soc-sign-bitcoinotc	5881	21492	6	2941	29	82.8	378283*	0.00	1384.4	378283*	0.00	**0.9**	**378283***	**0.00**	33.5	378283*	0.00	45.3	378283*
wiki-Vote	7115	100762	7	3558	36	**3.7**	**502576***	**0.00**	OT	632286	25.81	16.4	502576*	0.00	11.7	502576*	0.00	13.4	502576*
lasftm-asia	7624	27806	8	3812	38	OT	N/A	INF	OT	N/A	INF	**1.5**	**153512***	**0.00**	13.4	153512*	0.00	20.7	153512*
ca-HepTh	9875	25973	10	4938	49	OT	N/A	INF	OT	N/A	INF	OT	696761	0.33	**59.4**	**525179***	**0.00**	82.6	525179*
wiki-RfA	11380	181906	11	5690	57	7.3	789235*	0.00	683.8	789235*	0.00	**1.2**	**789235***	**0.00**	1.1	789923	0.09	1.3	789235*
ca-HepPh	12006	118489	12	6003	60	OT	447319	83.53	OT	489379	100.78	**2.4**	**243736***	**0.00**	1.2	243861	0.05	7.8	243736*
cit-HepTh	27769	352285	27	13885	139	OT	912209	97.56	OT	974319	111.01	OT	496698	0.08	243.6	478891	3.71	**1343.4**	**461744***

over $[0.3|V|, 0.7|V|]$. The partition is generated in the same way as in the SNAP instances. Terminals are inherited from the original graph, *i.e.*, $T_k = T \cap V_k$.

- **DIMACS11 Instances.** We collect 28 graphs from the 11th DIMACS Implementation Challenge (DIMACS11)[4]. It contains small-sized and large-sized graphs, from sparse to dense. These graphs are converted into PSTP instances in the same way as in the SteinLib instances.
- **RANDOM Instances** The random instances are synthetic SMP instances with a number of vertices ranging from 1000 to 10000. Let m be an integer, we denote by ran$m$ the instance of $1000 \times m$ vertices. The RANDOM instances are generated as follows: The density $|E|/|V|$ of ran$m$ is set to 5 and p is set to 3. The cardinality of V_k is 600 m, and each $|T_k|$ is fixed to 30. We first build a connected graph with 1000 m vertices and 5000 m edges by randomly traversing a complete graph until the required number of vertices and edges are visited and then drop the unvisited edges. Then all the sets V_k and T_k are generated in the same way as in SNAP. We create 100 instances for each ran$m$, with m ranging from 1 to 10, thus there are 1000 instances in RANDOM.
- **Wildlife Preservation Instances.** This benchmark instance originates from the real-world wildlife corridor design problem in Lai et al. (2011). All the graphs are grids representing the real maps. We generate these grid graphs in the same way as in Lai et al. (2011). We denote by grid$n$ the instance of a $n \times n$ grid graph, and p is set to 2 as in Lai et al. (2011). Each $|V_k|$ is generated randomly according to a uniform distribution over $[0.7|V|, 0.9|V|]$. Each $|T_k|$ is fixed at 9. Similar to the generation of RANDOM instances, we first build a complete $n \times n$ grid graph and then obtain the sets V_k and T_k in the same way as in SNAP. We create 100 instances for each grid$n$, with n ranging from 10 to 40 with a step of 5, thus there are 700 PSTP instances in this dataset.

Furthermore, for all instances, the vertex cost for each $u \in V$ is an integer generated randomly according to a uniform distribution over $[50, 1000]$. We also generate the vertex costs according to an asymmetric distribution. Based on preliminary tests, we observe that the distribution does not have a significant impact on solution times. Thus in the following experiments, all the tests are carried out on instances with vertex costs uniformly generated.

5.2. Experimental results

5.2.1. Results on snap, SteinLib and DIMACS11 instances

In what follows, we use MCF (Steiner) to denote the method by which we get a solution using CPLEX with the MCF (Steiner tree) formulation. We denote by VS-BC, VS-LB and VS-Hybrid the exact branch-and-cut algorithm, the local-branching-based heuristic, and the hybrid algorithm based on the VS formulation, respectively. We first test all five methods, *i.e.*, MCF, Steiner, VS-BC, VS-LB and VS-Hybrid, on non-random benchmark sets, *i.e.*, SNAP, DIMACS11 and SteinLib. In Table 2, 3 and 4, we show results for these instance sets, respectively.

In these tables, the first five columns show basic information about input instances. Specifically, columns $|V_k|$ and $|T_k|$ show the average size of all partitions and terminal sets, respectively. The column *time* reports running times measured in seconds, *OT* in the *time* column denotes that the corresponding exact algorithm fails to reach the optimality within the time limit. The *obj* column reports the objective values obtained by the corresponding method. An asterisk is added to the objective value if it is optimal. N/A indicates that no feasible solution is found. The *gap* column reports the relative gap between *obj* and the optimal solution value *opt*, *i.e.*, $gap = \frac{obj-opt}{opt} \times 100$. In case an optimal solution is not achieved by all the tested algorithms within the time limit, we resort to VS-Hybrid without time limit until the optimality is reached. However, *gap* can be INF if no feasible solution is returned. In the first line of each table, we show next to *Ins.* the number of instances in the corresponding set. Furthermore, next to each method's designation, we also show the number of instances for which an optimal solution is found. Easy instances that are solved by all algorithms in less than 1 second and difficult ones for which all algorithms fail to obtain a feasible solution within 1 hour are omitted. In bold are the best results with respect to the time used to reach optimality or the solution quality if the time limit is reached. In Table 2, the *gap* column for VS-Hybrid is omitted since this algorithm reaches optimality for every instance. In Table 4, the *gap* columns for VS-BC, VS-LB and VS-Hybrid are also omitted for the same reason.

From Tables 2 to 4, we can conclude that VS-Hybrid is the most efficient algorithm. It solves all the instances to optimality except cc7-3n and s5 from DIMAC11. Even for cc7-3n and s5, VS-Hybrid still hits the lowest objective values over all other algorithms. Notably, VS-Hybrid finds the optimal solution for cit-HepTh from SNAP, while none of the other algorithms can solve this instance.

If we only compare the exact algorithms MCF, Steiner and VS-BC, our VS-BC shows the best performances in terms of solution quality and running time. VS-BC can often optimally solve instances that MCF and Steiner cannot solve and the gaps obtained by VS-BC in solutions where that model does not reach optimality within the time limit dominate those of MCF and Steiner. VS-BC also runs up to two orders of magnitude faster than the other two algorithms. It is worth noting that VS-BC is particularly powerful on instances with a small number of vertices, *e.g.,* see Table 4, possibly owing to the fact that VS model has fewer vertex variables.

The heuristic algorithm VS-LB has higher possibilities of finding optimal solutions than MCF and Steiner, even if it cannot prove optimality. Indeed, it always outputs a solution of a small gap in hundreds

[4] http://dimacs11.zib.de/home.html

Table 3
Results for instances from DIMACS 11.

Ins.(22)	$\|V\|$	$\|E\|$	$\|P\|$	$\|V_k\|$	$\|T_k\|$	MCF(9)			Steiner(6)			VS-BC(16)			VS-LB(12)			VS-Hybrid(20)		
						Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap
rc04	121	197	2	72	44	0.6	33041*	0.00	12.8	33041*	0.00	**0.0**	**33041***	**0.00**	23.2	33041*	0.00	23.2	33041*	0.00
cc5-3n	243	1215	3	93	12	3.2	16696*	0.00	9.0	16696*	0.00	**0.8**	**16696***	**0.00**	1.5	16696*	0.00	1.8	16696*	0.00
rc05	247	486	4	148	51	5.7	49495*	0.00	213.9	49495*	0.00	**0.0**	**49495***	**0.00**	0.3	49495*	0.00	0.4	49495*	0.00
rt01	262	740	3	157	5	0.5	7563*	0.00	15.3	7563*	0.00	**0.0**	**7563***	**0.00**	1.3	7563*	0.00	1.6	7563*	0.00
cc9-2n	512	2304	2	264	32	OT	45225	0.34	OT	52891	17.35	**10.3**	**45073***	**0.00**	19.4	45078	0.01	23.8	45073*	0.00
cc6-3n	729	4368	2	280	33	1972.8	38782*	0.00	OT	408391	953.04	**3.5**	**38782***	**0.00**	53.8	38782*	0.00	89.3	38782*	0.00
s3	743	2947	4	240	50	45.3	64867*	0.00	532.3	64867*	0.00	**0.0**	**64867***	**0.00**	0.4	64867*	0.00	0.8	64867*	0.00
rt02	788	1938	4	472	22	7.1	16560*	0.00	33.7	16560*	0.00	**0.1**	**16560***	**0.00**	1.8	16559	0.00	2.6	16560*	0.00
cc3-10n	1000	13500	2	523	23	OT	27783	0.29	OT	31392	13.03	**12.3**	**27702***	**0.00**	86.9	27702*	0.00	103.6	27702*	0.00
cc10-2n	1024	5120	2	401	55	OT	67167	1.44	OT	69437	3.43	48.8	66214*	0.00	**36.8**	**66214***	**0.00**	50.3	66214*	0.00
cc3-11n	1331	19965	2	870	42	OT	63663	92.87	OT	82192	56.13	OT	33039	0.09	23.5	33137	0.39	**53.5**	**33008***	**0.00**
rc10	1572	3245	4	943	299	539.0	279592*	0.00	OT	283910	1.54	**0.1**	**279592***	**0.00**	1.7	279592*	0.00	1.8	279592*	0.00
rt03	1725	4092	2	1035	78	636.4	63088*	0.00	OT	82913	31.42	82.5	63088*	0.00	4.9	63153	0.10	**6.3**	**63088***	**0.00**
cc3-12n	1728	28512	2	1130	49	OT	709034	1967.03	OT	59234	72.68	OT	34643	0.99	53.4	35089	2.29	**103.6**	**34302***	**0.00**
cc11-2n	2048	11263	4	802	97	OT	216996	82.34	OT	382194	221.15	1205.0	119008*	0.00	23.5	119071	0.05	**348.5**	**119008***	**0.00**
cc7-3n[a]	2187	15308	2	842	82	OT	213969	97.68	OT	213969	97.68	OT	108264	0.02	**229.3**	**109232**	**0.01**	OT	109232	0.01
rc06	2502	6244	3	1501	59	OT	54079	0.06	OT	64231	18.84	**14.3**	**54047***	**0.00**	24.5	54047*	0.00	39.3	54047*	0.00
rc07	2740	6578	4	1644	106	OT	97172	0.09	OT	108283	11.53	**18.5**	**97085***	**0.00**	104.0	97085*	0.00	107.3	97085*	0.00
cc12-2n	4096	24574	2	2142	247	OT	N/A	INF	OT	N/A	INF	OT	284390	0.10	318.2	284701	0.25	**1489.4**	**283985***	**0.00**
s4	5202	20783	4	1682	344	OT	N/A	INF	OT	N/A	INF	1.7	447301*	0.00	**14.5**	**447301***	**0.00**	16.9	447301*	0.00
rt04	9469	22743	3	5681	12	OT	23942	5.27	OT	64981	185.72	OT	22745	0.01	592.3	24819	9.13	**1834.3**	**22743***	0.00
s5[b]	36415	145635	4	16554	2402	OT	N/A	INF	OT	N/A	INF	OT	13112345	80.01	425.7	7294831	0.14	OT	**7284814**	**0.01**

[a]The optimal value for cc7-3n is 108242, obtained by VS-Hybrid after 6 h.
[b]The optimal value for s5 is 7284323, obtained by VS-Hybrid after 8 h.

Table 4
Results for instances from SteinLib.

Ins.(50)	$\|V\|$	$\|E\|$	$\|P\|$	$\|V_k\|$	$\|T_k\|$	MCF(44)			Steiner(34)			VS-BC(50)		VS-LB(50)		VS-Hybrid(50)	
						Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Time	Obj	Time	Obj
i320–211	320	1845	4	152	16	5.6	21454*	0.00	25.0	21454*	0.00	**0.1**	**21454***	0.9	21454*	1.5	21454*
i320–212	320	1845	4	152	16	2.3	22393*	0.00	15.8	22393*	0.00	**0.1**	**22393***	0.5	22393*	0.5	22393*
i320–213	320	1845	4	152	16	5.4	22088*	0.00	34.5	22088*	0.00	**0.2**	**22088***	0.6	22088*	0.8	22088*
i320–214	320	1845	4	152	15	7.9	23282*	0.00	72.8	23282*	0.00	**0.4**	**23282***	4.4	23282*	5.0	23282*
i320–215	320	1845	4	152	15	2.2	20667*	0.00	14.0	20667*	0.00	**0.0**	**20667***	0.5	20667*	0.4	20667*
i320–311	320	1845	2	191	48	2.9	38153*	0.00	30.2	38153*	0.00	**0.0**	**38153***	0.3	38153*	0.2	38153*
i320–312	320	1845	2	191	51	3.0	36422*	0.00	49.8	36422*	0.00	**0.0**	**36422***	0.1	36422*	0.2	36422*
i320–313	320	1845	2	191	53	2.7	38019*	0.00	17.8	38019*	0.00	**0.0**	**38019***	0.3	38019*	0.2	38019*
i320–314	320	1845	2	191	50	3.0	38688*	0.00	52.8	38688*	0.00	**0.0**	**38688***	0.2	38688*	0.1	38688*
i320–315	320	1845	2	191	45	3.4	31644*	0.00	13.5	31644*	0.00	**0.0**	**31644***	0.2	31644*	0.2	31644*
mc11	400	760	3	208	114	50.9	107576*	0.00	321.3	107576*	0.00	**0.1**	**107576***	0.5	107576*	0.6	107576*
mc7	400	760	3	208	90	558.0	98192*	0.00	OT	108347	10.34	**1.4**	**98192***	17.2	98192*	18.8	98192*
mc8	400	760	3	164	79	27.0	92442*	0.00	OT	99324	7.44	**0.1**	**92442***	1.1	92442*	1.0	92442*
i640–101	640	960	3	386	17	153.5	28847*	0.00	OT	39245	36.05	**1.6**	**28847***	30.8	28847*	27.2	28847*
i640–102	640	960	3	386	14	3.1	25255*	0.00	23.7	25255*	0.00	**0.8**	**25255***	4.3	25255*	5.2	25255*
i640–103	640	960	3	386	16	5.4	33472*	0.00	129.7	33472*	0.00	**0.8**	**33472***	10.1	33472*	11.3	33472*
i640–104	640	960	3	386	15	3.3	25170*	0.00	23.0	25170*	0.00	**0.0**	**25170***	1.1	25170*	1.1	25170*
i640–105	640	960	3	386	16	103.4	29097*	0.00	643.3	29097*	0.00	**4.3**	**29097***	5.1	29097*	6.5	29097*
i640–203	640	960	3	247	22	20.3	36842*	0.00	126.1	36842*	0.00	**0.4**	**36842***	10.5	36842*	11.9	36842*
i640–204	640	960	3	247	20	13.8	41521*	0.00	136.3	41521*	0.00	**0.7**	**41521***	13.4	41521*	12.6	41521*
i640–205	640	960	3	247	19	1.0	33706*	0.00	8.9	33706*	0.00	**0.0**	**33706***	0.7	33706*	0.6	33706*
i640–303	640	960	4	337	84	21.7	120396*	0.00	185.4	120396*	0.00	**0.1**	**120396***	1.6	120396*	1.7	120396*
i640–304	640	960	4	337	90	18.4	122067*	0.00	185.6	122067*	0.00	**0.0**	**122067***	2.0	122067*	2.2	122067*
i640–305	640	960	4	337	84	12.5	112200*	0.00	69.4	112200*	0.00	**0.0**	**112200***	0.8	112200*	0.9	112200*
i640–031	640	1280	2	382	6	10.9	9072*	0.00	105.4	9072*	0.00	**1.9**	**9072***	25.3	9072*	29.3	9072*
i640–231	640	1280	3	317	27	601.8	43930*	0.00	OT	53948	22.80	**18.7**	**43930***	52.3	43930*	66.1	43930*
i640–232	640	1280	3	317	26	106.5	35821*	0.00	OT	38234	6.74	**1.4**	**35821***	23.2	35821*	22.8	35821*
i640–233	640	1280	3	317	26	15.5	42034*	0.00	91.4	42034*	0.00	**0.4**	**42034***	2.5	42034*	2.2	42034*
i640–234	640	1280	3	317	22	323.3	38844*	0.00	2658.1	38844*	0.00	**1.5**	**38844***	24.7	38844*	27.1	38844*
i640–235	640	1280	3	317	27	395.2	44730*	0.00	OT	49372	10.38	**2.4**	**44730***	66.5	44730*	65.5	44730*
i640–111	640	4135	3	386	14	263.3	16342*	0.00	OT	21348	30.63	**1.3**	**16342***	34.3	16342*	29.2	16342*
i640–112	640	4135	3	386	14	61.8	19063*	0.00	583.3	19063*	0.00	**0.7**	**19063***	16.1	19063*	17.1	19063*
i640–113	640	4135	3	386	16	79.3	16766*	0.00	2743.3	16766*	0.00	**1.5**	**16766***	18.4	16766*	22.7	16766*
i640–114	640	4135	3	386	14	651.3	19129*	0.00	OT	21341	11.56	**1.8**	**19129***	64.5	19129*	47.3	19129*
i640–115	640	4135	3	386	15	64.4	15184*	0.00	1345.3	15184*	0.00	**0.8**	**15184***	19.8	15184*	16.4	15184*
i640–211	640	4135	3	247	17	9.0	24344*	0.00	76.9	24344*	0.00	**0.2**	**24344***	4.6	24344*	5.2	24344*
i640–212	640	4135	3	247	20	531.7	28350*	0.00	OT	32834	15.82	**3.3**	**28350***	44.0	28350*	48.9	28350*
i640–213	640	4135	3	247	20	17.3	27758*	0.00	107.5	27758*	0.00	**0.3**	**27758***	2.7	27758*	4.2	27758*

(*continued on next page*)

Table 4 (*continued*).

Ins.(50)	$\|V\|$	$\|E\|$	$\|P\|$	$\|V_k\|$	$\|T_k\|$	MCF(44)			Steiner(34)			VS-BC(50)		VS-LB(50)		VS-Hybrid(50)	
						Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Time	Obj	Time	Obj
i640–311	640	4135	4	337	84	26.4	87240*	0.00	381.4	87240*	0.00	**0.1**	**87240***	0.8	87240*	0.8	87240*
i640–312	640	4135	4	337	88	25.2	87073*	0.00	639.8	87073*	0.00	**0.0**	**87073***	0.6	87073*	0.6	87073*
i640–313	640	4135	4	337	85	22.7	86523*	0.00	217.4	86523*	0.00	**0.0**	**86523***	0.5	86523*	0.6	86523*
i640–321	640	204480	4	337	160	OT	N/A	INF	OT	N/A	INF	**0.1**	**82487***	1.2	82487*	2.0	82487*
i640–322	640	204480	4	285	160	OT	N/A	INF	OT	N/A	INF	**0.1**	**83191***	1.7	83191*	1.2	83191*
i640–323	640	204480	2	389	160	OT	N/A	INF	OT	N/A	INF	**0.0**	**86631***	0.9	86631*	0.8	86631*
i640–324	640	204480	2	277	160	OT	N/A	INF	OT	N/A	INF	**0.0**	**82844***	0.5	82844*	0.7	82844*
i640–325	640	204480	3	402	160	OT	N/A	INF	OT	N/A	INF	**0.1**	**88321***	1.6	88321*	1.2	88321*
world666	666	221445	3	359	100	5.5	66456*	0	30.8	66456*	0.00	**0.1**	**66456***	1.0	66456*	1.0	66456*
w13c29	783	2262	3	384	196	249.7	200816*	0.00	2348.3	200816*	0.00	**0.0**	**200816***	0.8	200816*	0.9	200816*
w23c23	1081	3174	3	531	272	246.8	331285*	0.00	OT	374132	12.93	**0.0**	**331285***	0.8	331285*	0.8	331285*
w3c571	3997	10278	3	1964	1122	OT	N/A	INF	OT	N/A	INF	**0.8**	**1289131***	4.6	1289131*	6.5	1289131*

Table 5
Results for random graphs.

Ins.	MCF				Steiner				VS-BC				VS-LB				VS-Hybrid			
	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap
ran1	95	20.4	0.24	1.94	84	132.5	3.23	8.83	100	6.1	N/A	N/A	100	53.7	N/A	N/A	100	86.5	N/A	N/A
ran2	98	38.2	0.45	1.34	89	183.7	5.38	13.82	100	8.9	N/A	N/A	99	71.1	0.05	0.05	100	133.0	N/A	N/A
ran3	88	104.9	0.09	0.45	67	583.7	1.82	7.73	100	25.6	N/A	N/A	100	110.3	N/A	N/A	100	207.2	N/A	N/A
ran4	76	284.9	0.72	2.35	49	892.8	12.72	37.82	95	47.3	0.06	0.12	91	128.3	N/A	N/A	100	265.7	N/A	N/A
ran5	70	418.0	1.58	16.72	38	1739.8	8.82	21.73	96	84.0	0.05	0.25	97	194.1	0.01	0.05	100	345.9	N/A	N/A
ran6	70	829.7	2.73	9.65	23	2138.8	29.28	119.82	94	154.2	0.24	1.45	98	257.9	0.05	1.30	98	438.0	0.02	0.04
ran7	62	1642.5	4.09	19.81	12	1782.8	283.82	882.82	86	304.9	0.53	4.83	77	237.3	0.39	4.45	100	532.2	N/A	N/A
ran8	45	2045.1	10.72	30.71	17	1642.3	192.83	1137.7	72	757.8	0.94	5.96	73	298.2	1.53	5.92	96	919.5	0.05	1.40
ran9	30	2552.6	12.52	192.63	8	2523.7	478.62	3373.8	47	1479.3	0.9	13.71	41	435.0	2.09	9.30	90	1514.3	0.04	0.43
ran10	15	3205.8	37.7	387.45	3	2384.7	899.83	7392.83	70	2190.2	1.58	24.58	57	505.6	8.52	15.51	91	1891.2	0.02	0.90

of seconds, even for hard instances. For ca-HepTh from SNAP, it spends less than 60 seconds to find the optimal solution, while MCF, Steiner and VS-BC fail in even one hour.

5.2.2. Results on random graph

In Table 5, we report results on RANDOM dataset, using 100 instances for each ran$m$. Column *Slv.* reports the number of instances that are solved to optimality within one hour. Column *time* indicates the average running time of the algorithm on those instances for which an optimal solution is obtained within one hour. Column *a-gap* reports the average gap for those instances where the optimal solution is not achieved and *m-gap* reports the maximum gap among them. If for a certain ran$m$, all 100 instances are optimally solved, gaps are indicated by *N/A*.

Results in Table 5 confirm that VS-Hybrid performs better than its competitors, with respect to the number of solved instances and solution quality of instances where the optimal solution is not obtained. It optimally solved more than 95% of all the 1,000 samples. For instances where the optimal solution is not achieved, VS-Hybrid obtains the smallest gaps. Specifically, for ran10 which has 10,000 vertices, VS-Hybrid successfully solves 91 out of 100 large instances with 10,000 vertices. As for the other algorithms, VS-BC performs better than MCF and Steiner, in terms of solution quality and running time.

We further investigate the performance of VS with different $|T_k|$ and p. In Fig. 4(a), we show the average running time of VS-BC with respect to different $|T_k|$ on 100 RANDOM instances where $|V| = 1000$, $|E| = 5000$, $|V_k| = 600$ and $p = 3$. In Fig. 4(b), we show the average running time of VS-BC with respect to different p, also, on 100 RANDOM instances with $|V| = 1000$, $|E| = 5000$, $|V_k| = 600$ and $|T_k| = 30$. Clearly, the running time of VS-BC reaches a maximum when $|T_k|$ is around 120, but drops when $|T_k|$ is far from 120. Also, the running time of VS-BC grows exponentially as the number of partitions increases.

5.2.3. Results on wildlife preservation instances

In Table 6, we report the results obtained on grid graphs. In Lai et al. (2011), Lai et al. solve the PSTP in grid graphs with 25 × 25 vertices. Following their work, we also test grid graphs of vertex numbers within 40 × 40. In Table 6, we show the same information as in Table 5.

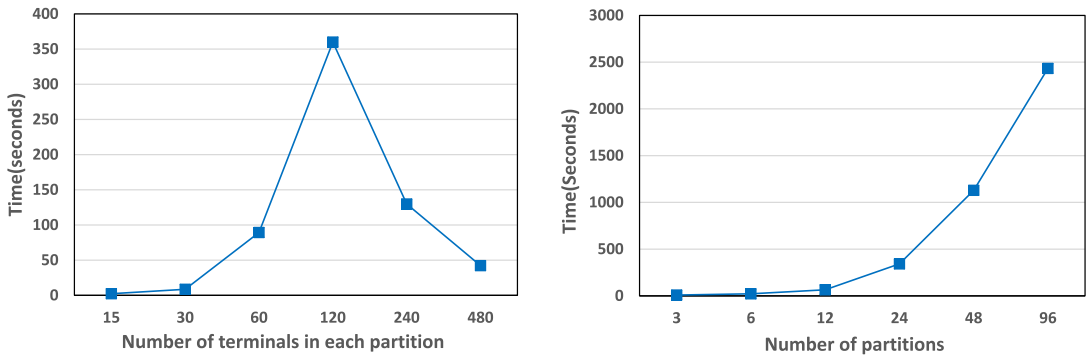
As we can see, VS-Hybrid still performs very well in terms of the number of solved instances and optimality gap for the instances where no proven optimum is known. However, it is surprising that MCF is another strong solver for these instances. Indeed, only MCF and VS-Hybrid can solve more than half of the grid35 and grid40 instances.

5.2.4. Evaluating the strength of additional inequalities

In Section 3.3, we introduce additional inequalities to tighten the initial VS model. In this section, we investigate the impact of these additional inequalities by comparing the performance of our VS model initialized with and without these inequalities. We first conduct comparison experiments for the neighbor inequalities, then the 2D grid inequalities.

To evaluate the impact of the neighbor inequalities on our VS model, we select one instance from each of the five data sets: wiki-RfA from SNAP, cc10-2n from DIMACS11, i640-231 from SteinerLib, one instance of ran3 from RANDOM, one instance of grid20 from wildlife preservation. For each instance, we first compute and store all the neighbor inequalities, *i.e.*, inequalities of constraints (5), (8) and (9). Inequalities of constraint (5) are first generated, then constraint (8), and finally constraint (9). These inequalities are arranged according to the order of their generation. Then we run the branch-and-cut algorithm initialized with objective function (1), constraints (2), (4), and the first $10n$ percentage of the ordered neighbor inequalities, where $n = 0, 1, \ldots, 10$. If the graph is a 2D grid graph, the 2D grid inequalities are also included.

To quantify the level of relaxation, a new indicator *lp-gap* is introduced. Recall that opt is the optimal objective value of the ILP. Let $lopt$ be the optimal objective value of the linear relaxation of the ILP. The quality of the lower bound obtained by the ILP relaxation can be evaluated by the relative gap between opt and $lopt$, that is, $lp\text{-}gap = \frac{opt - lopt}{opt} \times 100$. In Fig. 5, we report the *lp-gap* of VS-BC with respect to different percentages of total neighbor inequalities on the five instances. We show below the caption of each instance the number of total neighbor inequalities. In Fig. 6, we report the time used by VS-BC to achieve an optimal solution, with respect to different percentages



(a) Average running time of VS-BC for different terminal set size (b) Average running time of VS-BC for different partition numbers

Fig. 4. Average running time of VS-BC.

Table 6
Results for grid graphs.

Ins.	MCF				Steiner				VS-BC				VS-LB				VS-Hybrid			
	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap	Slv.	Time	a-gap	m-gap
grid10	100	0.5	N/A	N/A	100	3.3	N/A	N/A	100	0.8	N/A	N/A	100	2.3	N/A	N/A	100	4.5	N/A	N/A
grid15	100	3.0	N/A	N/A	99	29.9	0.32	0.32	100	2.3	N/A	N/A	81	9.5	0.01	0.39	100	34.5	N/A	N/A
grid20	98	40.7	0.04	0.23	74	372.5	1.38	7.29	97	12.3	0.44	2.45	92	18.3	0.35	0.96	100	50.3	N/A	N/A
grid25	91	231.4	0.42	1.94	58	1734.8	23.42	134.92	90	90.0	2.34	6.43	68	82.4	0.84	2.34	93	183.5	0.73	1.9
grid30	86	658.7	1.23	3.45	34	2633.3	82.45	573.72	60	669.4	3.56	23.34	35	294.5	1.53	4.82	86	582.8	1.92	2.75
grid35	73	1123.5	0.94	4.38	15	2924.0	282.65	921.82	52	1342.0	11.39	39.38	20	523.7	2.34	7.48	73	1504.7	1.04	3.82
grid40	59	2703.4	2.34	8.92	11	2722.7	273.31	1382.73	30	2543.0	13.44	93.48	14	594.4	3.57	17.34	61	2742.9	0.22	5.43

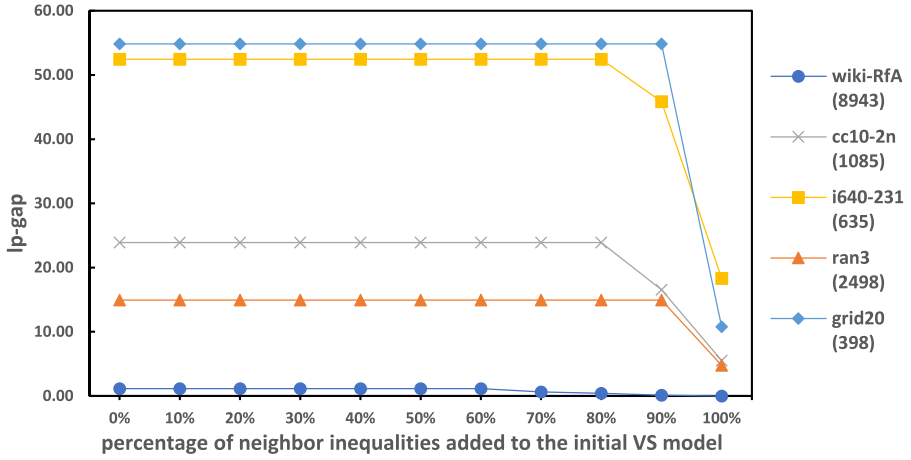


Fig. 5. The linear relaxation gaps of VS-BC for different percentages of neighbor inequalities added to the initial VS model.

of total neighbor inequalities. As we can see in Fig. 5, for each instance, there is a clear decrease in the *lp-gap* when more than 90% of the neighbor inequalities are added to the initial VS model. Before the clear decrease, the *lp-gap* tends to remain unaffected by the number of added neighbor inequalities. The time to reach optimality, as shown in Fig. 6, also demonstrates an observable decrease when more than 30% of the neighbor inequalities are added. A prominent example is the running time of ran3, which drops from 217 seconds to 40 second when 30% of the neighbor inequalities are added.

Next, we evaluate the strength of 2D grid inequalities. In Table 7 we compare the results of VS-Hybrid for all instances in wildlife preservation data set in two settings: the initial VS model (objective function (1), constraints (2), (4), and neighbor inequalities (5), (8) and (9)) of the underlying branch-and-cut algorithm incorporates the 2D grid inequalities and that does not incorporate the 2D grid inequalities. Note that the solution of the linear relaxation of the ILP is sometimes an integer solution, which is, of course, an optimal solution of the ILP. Thus the number of the linear relaxations that obtain an integer solution can also reflect the strength of the ILP relaxation and the column *lp-slv* reports this number in Table 7. The columns *nodes* and *cuts* report the average number of explored nodes and the average number of generated cuts, respectively. As we can see in Table 7, after adding the 2D grid inequalities, both computational times and gaps are improved. Notably, a speedup factor of 4 is often observed regarding the time to reach optimality. From the last two lines of Table 7, we can see that the times get larger with 2D grid inequalities than without them. We should mention that this is due to the fact that more instances are solved with 2D grid inequalities, and those that cannot be optimally solved without 2D grid inequalities are difficult, so the average time with 2D grid inequalities gets larger.

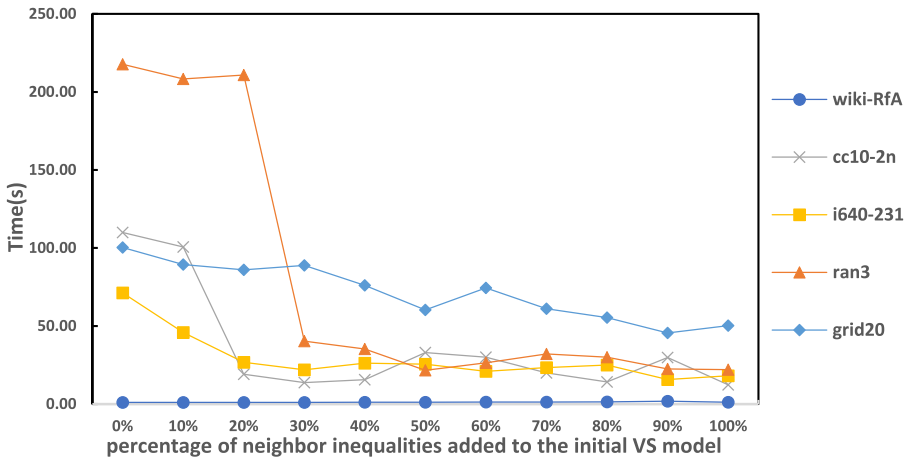


Fig. 6. The time to reach optimality of VS-BC for different percentages of neighbor inequalities added to the initial VS model.

Table 7
Comparison results for 2D grid inequalities.

Ins.	VS-Hybrid with 2D grid inequalities								VS-Hybrid without 2D grid inequalities							
	LPgap	lp-slv	Slv.	Time	a-gap	m-gap	Nodes	Cuts	lp-gap	lp-slv	Slv.	Time	a-gap	m-gap	Nodes	Cuts
grid10	8.99	50	100	4.5	N/A	N/A	21	264	33.38	30	100	21.9	N/A	N/A	46	814
grid15	8.40	31	100	34.5	N/A	N/A	38	373	29.14	16	100	256.9	N/A	N/A	197	2352
grid20	7.66	26	100	50.3	N/A	N/A	114	771	30.74	10	99	234.9	0.52	0.52	309	1877
grid25	12.66	14	93	183.5	0.73	1.9	323	2234	35.14	4	83	767.6	2.86	8.40	1139	3726
grid30	15.48	11	86	582.8	1.92	2.75	852	4028	34.40	3	58	1156.7	9.58	15.26	7081	30532
grid35	19.58	8	73	1504.7	1.04	3.82	1741	13629	46.46	1	59	1129.8	7.29	15.27	4543	25536
grid40	25.23	4	61	2742.9	0.22	5.43	3393	21128	74.25	0	34	2005.5	2.97	18.80	6881	65075

6. Conclusions and future work

The PSTP is an important network design problem in computational sustainability, network protection and social networks. In general, this problem is computationally challenging, even under quite restrictive assumptions. In the paper, we mainly investigated practically efficient integer programming models and algorithms for the problem.

We first discussed existing Integer Linear Program (ILP) formulations for the PSTP. Then we suggested a novel vertex-separator (VS) based ILP model with much fewer variables than existing ILPs. We also showed additional valid neighbor inequalities and new vertex-separators inequalities in 2D grid graphs to tighten the VS model.

Afterward, we developed three algorithms based on the VS model for solving the PSTP in general graphs. The first one is an exact branch-and-cut algorithm. Due to the exponential number of inequalities in VS model, an effective separation oracle is given to find violated inequalities in the branch-and-cut. The second algorithm is a fast local-branching-based heuristic, which utilizes the VS model for searching neighbor solutions. The last algorithm, which combines exact branch-and-cut and local-branching heuristics, mixes advantages of both methods.

By thorough experiments, we showed that our algorithms based on the VS model outperform existing methods of the literature in terms of solution quality and running time. Our methods can solve large-sized instances of more than ten thousand vertices. Computational results also indicate that our hybrid algorithm is very efficient in solving different types of instances.

Future work can be conducted from both application and algorithmic perspectives. On the one hand, the PSTP has a wide range of existing and potential applications, so it would be interesting to explore the algorithms in different domains. On the other hand, it is worth considering new algorithmic techniques like machine learning aided search or parallelization in the algorithms.

CRediT authorship contribution statement

Mengfan Ma: Software, Validation, Data curation, Writing – original draft, Writing – review & editing. **Ziyang Men:** Software, Validation, Data curation. **André Rossi:** Methodology, Investigation, Writing – review & editing. **Yi Zhou:** Validation, Writing – review & editing, Supervision, Funding acquisition. **Mingyu Xiao:** Supervision, Project administration.

Data availability

Data will be made available on request.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China under grant nos. 61802049, 61972070 and 62172077.

Appendix A. The framework of our branch-and-cut method

The algorithmic framework of our branch-and-cut method for the PSTP is depicted in Algorithm 6.

Appendix B. Other ILPs for the PSTP

B.1. Single-commodity flow model

In the SCF formulation for the PSTP, we use a single-commodity network flow to encode the connectivity (Conrad et al., 2007, Dilkina and Gomes, 2010). The formulation is proposed in Conrad et al. (2007) to model another problem in graphs. First, we introduce a source vertex s. Then for each $k \in [p]$, we convert $G_k = (V_k, E_k)$ into a flow network by the following operations:

Algorithm 6: Branch-and-cut framework

Input: A PSTP instance $I = (G, c, \mathcal{V}, \mathcal{T}, p)$, time limit $BCTimeLim$, parameters (ϵ_1, α_1) and (ϵ_2, α_2)
Output: A solution for instance I
Initialize: upper bound $\overline{z} = +\infty$, incumbent solution $\boldsymbol{x}^* \leftarrow (1, \cdots, 1)$, set of active subproblems $L \leftarrow \emptyset$, add to L the initial partial VS model, denoted by ILP_0 /* the initial partial VS model consists of objective function (1), constraints, (2), (4), and the neighbor inequalities constraints (5), (8) and (9). When G is a 2D grid graph, constraint (10) is also included. */

1 **while** ($L \neq \emptyset$) *and* (*running time* $\leq BCTimeLim$) **do**
2 Select and remove ILP_i from L
3 Solve the LP relaxation of ILP_i, denoted by LP_i /* let $\boldsymbol{x}$ be the solution of LP_i */
4 **if** (IP_i *is infeasible) or* ($\sum_{u\in V} c_u x_u > \overline{z}$) **then**
5 continue
6 **else**
  /* let IntegratedSeparation be Algorithm 3 */
7 $Q \leftarrow$IntegratedSeparation$(I, \boldsymbol{x}, (\epsilon_1, \alpha_1), (\epsilon_2, \alpha_2))$
8 **if** $Q \neq \emptyset$ **then**
9 Add Q to LP_i and go to line 3
10 **else**
11 **if** ($\sum_{u\in V} c_u x_u \leq \overline{z}$) *and* ($\boldsymbol{x}$ *is an integer solution*) **then**
12 $\overline{z} \leftarrow \sum_{u\in V} c_u x_u$, $\boldsymbol{x}^* \leftarrow \boldsymbol{x}$
13 Split LP_i into subproblems and add them to L
14 **end**
15 **end**
16 **end**
17 **end**
18 **return** $\boldsymbol{x}^*$

- Each undirected edge (u, v) in G_k is replaced by two directed edges (u, v) and (v, u), called the directed edge set A_k
- Inject a flow of size $|V_k|$ into s, call the flow the *k-th flow*.
- An arbitrary terminal vertex $r_k \in T_k$ is chosen as *root* r_k, and a directed edge (s, r_k) is defined to insert the flow into G_k.

For each $k \in [p]$, the source vertex s is associated with variable y_s^k to represent the eventual residual kth flow in s. Two types of variables are associated with each vertex. For each $k \in [p]$ and each vertex $u \in V_k$, a binary variable y_u^k is introduced to indicate whether or not the kth flow traverses vertex u. For each $u \in V$, the other binary variable x_u indicates whether $u \in V$ is in the final solution.

Besides, for each $k \in [p]$ and for each directed edge $(u, v) \in (u, v) \in A_k \cup \{(s, r_k)\}$, there is a non-negative *flow variable* f_{uv}^k to indicate the amount of the kth flow from u to v. The whole SCF formulation is shown as follows:

$$\text{minimize} \sum_{u\in V} c_u x_u \tag{B.1}$$

$$y_t^k = 1 \qquad \forall k \in [p], \forall t \in T_k \tag{B.2}$$

$$x_u \geq y_u^k \qquad \forall k \in [p], \forall u \in V_k \tag{B.3}$$

$$f_{sr_k}^k + y_s^k = |V_k| \qquad \forall k \in [p] \tag{B.4}$$

$$f_{uv}^k \leq |V_k| y_v^k \qquad \forall k \in [p], \forall (u, v) \in A_k \tag{B.5}$$

$$\sum_{v:(v,u)\in G_k} f_{vu}^k = y_u^k + \sum_{v:(u,v)\in G_k} f_{uv}^k \quad \forall k \in [p], \forall u \in V_k \tag{B.6}$$

$$\sum_{u\in V_k} y_u^k = f_{sr_k}^k \qquad \forall k \in [p] \tag{B.7}$$

$$y_u^k \in \{0, 1\} \qquad \forall k \in [p], \forall u \in V_k \tag{B.8}$$

$$y_s^k \in [0, |V_k|] \qquad \forall k \in [p] \tag{B.9}$$

$$x_u \in \{0, 1\} \qquad \forall u \in V \tag{B.10}$$

$$f_{uv}^k \geq 0 \qquad \forall k \in [p], \forall (u, v) \in A_k \cup \{(s, r_k)\} \tag{B.11}$$

Constraint (B.2) ensures that for each $k \in [p]$, each terminal in V_k must act as a sink for the kth flow. Constraint (B.3) establishes the relationship between x_u and y_u^k, i.e., $x_u = 1$ if there exists $k \in [p]$ such that $y_u^k = 1$. Constraint (B.4) states, for each kth flow, that the residual flow in s plus the flow injected into the network of G_k is equal to the amount of flow injected into s. For the kth flow, it is enforced by constraint (B.5) that each of the vertices with a positive incoming flow retains one unit of flow, i.e., $(f_{uv}^k > 0) \Rightarrow (y_v^k = 1)$, $\forall (u, v) \in A$. The flow conservation is modeled in constraint (B.6). Finally, constraint (B.7) enforces that the kth flow absorbed by vertices in V_k equals to the flow injected into the G_k. This encoding requires $O(p(|V| + 2|E|))$ variables, including $O(p|V|)$ binary variables and $O(p(|E|))$ continuous variables. The number constraints is also $O(p(|V| + 2|E|))$.

B.2. Multi-commodity flow model

With the SCF, connectivity within T_k is enforced through a single flow in each G_k. Lai et al. (2011) propose a Multi-Commodity Flow formulation(MCF), where the key difference is that they enforce the connectivity of T_k in G_k by associating a separate commodity with each vertex in T_k. For each $k \in [p]$, one arbitrary terminal vertex $r_k \in T_k$ is chosen as *root*, then the problem can be modeled as finding $|T_k|-1$ paths from r_k to each of the other terminals in $T_k \setminus \{r_k\}$. There is one unit of flow from the root to each other terminals in V_k. We call the flow from r_k to some other terminal $t \in T_k \setminus \{r_k\}$ the (k, t)-*flow*. For each (k, t)-flow and each $(u, v) \in A_k$, a continuous variable f_{uv}^{kt} is introduced to represent the (k, t)-flow from u to v.

Two types of variables are associated with each vertex in V_k. For each (k, t)-flow, and each $u \in V_k$, a continuous variable y_u^{kt} is associated with u to indicate the amount of incoming (k, t)-flow into u, as shown in constraint (B.20); for each $u \in V$, a binary variable x_u indicates whether $u \in V$ is in the final solution. Although continuous, y_u^{kt} can be seen as an indicator for whether the (k, t)-flow traverses vertex u: $y_u^{kt} > 0$ indicates that u is traversed by the (k, t)-flow, otherwise not. $x_u = 1$ indicates that u is in the final solution set, i.e., u is traversed by at least one (k, t)-flow. The MCF model is shown as follows:

$$\text{minimize} \sum_{u\in V} c_u y_u \tag{B.12}$$

$$x_u \geq y_u^{kt} \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\}, \forall u \in V_k \tag{B.13}$$

$$\sum_{u:(u,r_k)\in A_k} f_{ur_k}^{kt} = 0 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\} \tag{B.14}$$

$$\sum_{u\in V_k:(r_k,u)\in A} f_{r_k u}^{kt} = 1 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\} \tag{B.15}$$

$$y_{r_k}^{kt} = 1 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\} \tag{B.16}$$

$$\sum_{u:(u,t)\in A_k} f_{ut}^{kt} = 1 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\} \tag{B.17}$$

$$\sum_{u:(t,u)\in A_k} f_{tu}^{kt} = 0 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\} \tag{B.18}$$

$$y_t^{kt} = 1 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\} \tag{B.19}$$

$$\sum_{v:(v,u)\in A_k} f_{vu}^{kt} = y_u^{kt} \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\}, \forall u \in V_k \tag{B.20}$$

$$\sum_{v:(u,v)\in A_k} f_{uv}^{kt} = y_u^{kt} \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\}, \forall u \in V_k \tag{B.21}$$

$$y_u^{kt} \geq 0 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\}, \forall u \in V_k \tag{B.22}$$

Table C.1
Tuning of ϵ and α for Algorithm 1.

ϵ	0.001	0.1	0.2	0.4	0.8
α	Time/Nodes/Cuts	Time/Nodes/Cuts	Time/Nodes/Cuts	Time/Nodes/Cuts	Time/Nodes/Cuts
1	1.00/1.00/1.00	1.29/1.16/1.03	1.23/1.23/0.95	1.42/2.92/0.99	1.21/3.99/1.01
5	0.91/0.93/1.28	0.94/0.98/1.37	0.91/0.92/1.37	1.03/2.5/1.35	1.14/3.93/1.45
10	0.83/0.75/1.58	0.87/1.08/1.65	0.89/0.96/1.34	0.74/2.18/1.33	0.92/3.33/1.56
20	0.72/0.78/1.78	0.95/0.82/1.58	0.8/0.98/1.72	1.07/1.95/1.65	1.20/3.62/1.72
∞	**0.59**/0.78/1.73	0.71/0.93/1.75	0.8/0.93/1.66	1.02/2.4/1.67	1.51/4.86/1.82

$$x_u \in \{0,1\} \quad \forall u \in V \tag{B.23}$$

$$f^{kt}_{uv} \geq 0 \qquad \forall k \in [p], \forall t \in T_k \setminus \{r_k\}, \forall (u,v) \in G_k \tag{B.24}$$

Constraint (B.13) states that vertex u will be selected in the solution if it is traversed by at least one (k,t)-flow. Constraint (B.14) and (B.15) force r_k to be the source of each (k,t)-flow. Correspondingly, constraints (B.17) and (B.18) enforce t to be the sink of each (k,t)-flow. The flow conservation of intermediate vertices in each (k,t)-flow is given by constraints (B.20) and (B.21). Together with constraints (B.16) and (B.19), the fact that r_k and each other terminal in T_k must be chosen is captured.

MCF formulation requires $O(p|T|(|V|+2|E|))$ variables, including $O(p|T|(|V|+2|E|))$ continuous variables and $O(|V|)$ binary variables. The number of constraints is $O(p|T||V|)$. The number of variables and constraints in MCF is considerably more than in SCF. But as shown in Lai et al. (2011), MCF experimentally outperforms SCF. Theoretically, Taccari showed that for the longest path problem, MCF is tighter than SCF in Taccari (2016).

B.3. Steiner tree model

As suggested by the MCF formulation, to ensure connectivity within T_k in each G_k one may enforce that there exists a path from a root $r_k \in T_k$ to each other terminals in G_k(Dilkina and Gomes, 2010). The Steiner-tree ILP formulation for the PSTP is stated as:

$$\text{minimize} \sum_{u \in V} c_u x_u \tag{B.25}$$

$$x_u \geq \sum_{v:(v,u)\in G_k} z^k_{vu} \qquad \forall k \in [p], \forall u \in V \setminus \{r_k\} \tag{B.26}$$

$$\sum_{v:(v,u)\in G_k} z^k_{vu} \leq 1 \qquad \forall k \in [p], \forall u \in V_k \tag{B.27}$$

$$\sum_{u:(u,t)\in G_k} z^k_{ut} = 1 \qquad \forall k \in [p], \forall t \in T_k \tag{B.28}$$

$$\sum_{(u,v)\in A_k : v\in S, u\in V_k\setminus S} z^k_{uv} \geq \sum_{u:(u,w)\in A_k} z^k_{uw} \quad \forall k \in [p], \forall S \subseteq V_k \setminus \{r_k\}, \forall w \in S \tag{B.29}$$

$$z^k_{uv} + z^k_{vu} \leq 1 \qquad \forall k \in [p], \forall (u,v) \in A_k, u \neq r_k, v \neq r_k \tag{B.30}$$

$$x_u \in \{0,1\} \qquad \forall u \in V \tag{B.31}$$

$$z^k_{uv} \in \{0,1\} \qquad \forall k \in [p], \forall (u,v) \in A_k \tag{B.32}$$

In this formulation, we explicitly model the selection of edges in each G_k as binary variables. We insist that we select a set of vertices and edges in G_k such that there is a single path from the root to each selected vertex in G_k. In other words, we impose stronger constraints than necessary while preserving all feasible solutions in terms of subset for vertices that induce a connected subgraph in G_k. In effect, we enforce the connectivity constraints by adding constraints that ensure that we select edges that form a (Steiner) tree for each T_k in G_k, thus, a Steiner forest considering there are multiple partitions. This can be done by the generalized cut constraints. Several studies on Steiner tree problem variants have shown that often directed edge models are better than undirected ones in solving Steiner Tree problems (e.g., Ljubic et al., 2005 and Leggieri et al., 2012). Thus for each $k \in [p]$, we again replace each undirected edge $(u,v) \in E_k$ with two directed edges (u,v) and (v,u), call the directed edge set A_k. For each vertex $u \in V$, a binary variable x_u is introduced to indicate whether u is selected in the solution. For each $k \in [p]$ and each $(u,v) \in A_k$, a binary arc variable z^k_{uv} is introduced to indicate whether arc (u,v) is used for the connectivity of T_k in G_k.

We may also need a binary variable x^k_u as in SCF, to indicate whether vertex u is used for the connectivity of T_k in G_k. However, we can avoid explicitly including such variables, as these decisions can be inferred from the values of the arc variables y^k_{uv}: Each vertex used for the connectivity of T_k in G_k has exactly one incoming edge of the same use. That is, we have $x^k_u = \sum_{v:(v,u)\in A_k} y^k_{vu}$. The relationship the binary variable x_u and y^k_{vu} is stated by constraint (B.26). To enforce the directed tree property, each non-root vertex is allowed have at most one incoming edge in the Steiner tree of G_k (constraint (B.27)). Constraint (B.28) enforces that each terminal vertex should be selected in Steiner tree of G_k. Connectivity of T_k in each G_k is enforced through generalized cut constraints over each G_k (constraint (B.29)). We also include constraint (B.30), which strengths the formulation by enforcing that each edge is used at most one direction for the Steiner tree in G_k.

The number of variables in the Steiner-tree model is $O(p|E|+|V|)$, which are all binary variables. The number of constraints is $O(p|V|2^{|V|})$. Given the exponential number of generalized cut constraint (B.29), we also solve the model in a branch-and-cut framework and separate the generalized cut constraint by a minimum cut algorithm, as stated in Dilkina and Gomes (2010) and Taccari (2016).

Appendix C. Tuning parameters

In this section, we conduct parameter tuning procedures for our separation oracle and our local-branching-based heuristic algorithm.

C.1. Tuning parameters of separation oracle

In Section 4.1.3, we use ϵ and α in the separation algorithm. ϵ indicates the minimum violation of the separated inequalities, and α indicates the maximum number of violated inequalities that are separated.

In Table C.1 to C.2, we summarize a tuning procedure that is carried out on a subset of 100 random graphs with 2000 vertices to identify the best ϵ and α for Algorithm 1 and 2, respectively. We report the geometric mean of the time to optimality, the number of nodes, and the number of added cuts. The values are normalized, for each instance, with respect to the results with $\epsilon = 0.001$ and $\alpha = 1$. In bold is the fastest time for each algorithm.

C.2. Tuning parameters in local branching

We now tune parameters in the local-branching heuristic algorithm, i.e., Algorithm 4. Table C.3 summarizes these parameters, where LB and UB are the lower bound and upper bound for the parameters, respectively.

To simplify the tuning procedure, we let $LBMaxIter = (r_{max} - r_{min})/r_\delta$. Thus total time is upper bounded by $LBMaxIter \times LBTimeLim$. It is suggested in Fischetti et al. (2017), Fischetti and Monaci (2014) that a small radius is preferred for local-branching algorithm. We thus set radius $r_{min} = 10$ and $r_{max} = 30$.

Table C.2
Tuning of ϵ and α for Algorithm 2.

ϵ	0.001	0.1	0.2	0.4	0.8
α	Time/Nodes/Cuts	Time/Nodes/Cuts	Time/Nodes/Cuts	Time/Nodes/Cuts	Time/Nodes/Cuts
1	1.00/1.00/1.00	1.1/1.22/1.05	1.04/1.41/0.97	1.18/3.33/1.01	1.08/4.56/1.03
5	0.81/1.07/1.36	0.84/1.11/1.40	0.82/1.05/1.40	0.92/2.86/1.38	1.02/4.49/1.48
10	0.74/1.17/1.56	0.78/1.30/1.61	0.62/1.10/1.37	1.04/2.49/1.36	0.96/3.89/1.46
20	0.65/0.89/1.76	0.65/0.93/1.62	0.72/1.14/1.72	**0.61**/3.06/1.68	0.69/4.13/1.76
∞	0.97/0.95/1.74	1.03/1.06/1.74	1.04/0.98/1.70	0.93/2.74/1.70	1.35/5.55/1.86

Table C.3
Ranges of parameters in VS-LB.

Parameter	LB	UB	Description
r_{min}	1	r_{max}	minimum radius
r_{max}	r_{max}	$\|V\|$	maximum radius
r_δ	1	$r_{max} - r_{min}$	radius step
$LBMaxIter$	1	$\frac{r_{max}-r_{min}}{r_\delta}$	maximum iteration
$LBTimeLim$	0	∞	time limit

Table C.4
Local branching parameter tuning.

LBTimeLim	LBMaxIter	Time
5	1	1.00
	5	2.10
	10	1.14
10	1	0.63
	5	1.74
	10	**0.08**
20	1	0.30
	5	1.67
	10	2.03
50	1	2.18
	5	1.06
	10	0.37

In Table C.4, we show the tuning results on the same set of instances used in the tuning for separation oracles in Appendix C.1. For separation oracle, we use $\epsilon = 0.001$ and $\alpha = \infty$ for Algorithm 1, and $\epsilon = 0.4$ and $\alpha = 20$ as shown in Appendix C.1 We report the geometric mean of the running times of Algorithm 4. The values are normalized with respect to the results with $LBTimeLim = 5$ and $LBMaxIter = 1$. The minimum time is in bold.

References

Ahn, Yong-Yeol, Bagrow, James P., Lehmann, Sune, 2010. Link communities reveal multiscale complexity in networks. Nature 466 (7307), 761–764.

Alagador, Diogo, Trivino, Maria, Cerdeira, Jorge Orestes, Brás, Raul, Cabeza, Mar, Araújo, Miguel Bastos, 2012. Linking like with like: optimising connectivity between environmentally-similar habitats. Landsc. Ecol. 27 (2), 291–301.

Ali, Arshad, Ikpehai, Augustine, Adebisi, Bamidele, Mihaylova, Lyudmila, 2016. Location prediction optimisation in WSNs using Kriging interpolation. IET Wirel. Sens. Syst. 6 (3), 74–81.

Bley, Andreas, Ljubić, Ivana, Maurer, Olaf, 2017. A node-based ILP formulation for the node-weighted dominating Steiner problem. Networks 69 (1), 33–51.

Brás, Raul, Cerdeira, J. Orestes, 2015. Computational comparison of algorithms for a generalization of the node-weighted Steiner tree and forest problems. In: Operational Research. Springer, pp. 67–83.

Brás, Raul, Cerdeira, J. Orestes, Alagador, Diogo, Araújo, Miguel B., 2013. Linking habitats for multiple species. Environ. Model. Softw. 40, 336–339.

Buchanan, Austin, Wang, Yiming, Butenko, Sergiy, 2018. Algorithms for node-weighted steiner tree and maximum-weight connected subgraph. Networks 72 (2), 238–248.

Conrad, Jon, Gomes, Carla P., Van Hoeve, Willem-Jan, Sabharwal, Ashish, Suter, Jordan, 2007. Connections in networks: Hardness of feasibility versus optimality. In: Proceedings of CPAIOR. Springer, pp. 16–28.

Dilkina, Bistra, Gomes, C., 2009. Wildlife corridor design: Connections to computer science. In: Proceedings of the Workshop on Constraint Reasoning and Optimization for Computational Sustainability. CROCS-09, Lisbon: CP-09, pp. 3–4.

Dilkina, Bistra, Gomes, Carla P., 2010. Solving connected subgraph problems in wildlife conservation. In: Proceedings of CPAIOR. Springer, pp. 102–116.

Dilkina, Bistra, Lai, Katherine, Le Bras, Ronan, Xue, Yexiang, Gomes, Carla, Sabharwal, Ashish, Suter, Jordan, McKelvey, Kevin, Schwartz, Michael, Montgomery, Claire, 2013. Large landscape conservation—synthetic and real-world datasets. In: Proceedings of AAAI. AAAI, pp. 1–4.

Ding, Bolin, Yu, Jeffrey Xu, Wang, Shan, Qin, Lu, Zhang, Xiao, Lin, Xuemin, 2007. Finding top-k min-cost connected trees in databases. In: Proceedings of ICDE. IEEE, pp. 836–845.

Dreyfus, Stuart E., Wagner, Robert A., 1971. The Steiner problem in graphs. Networks 1 (3), 195–207.

Faloutsos, Christos, McCurley, Kevin S., Tomkins, Andrew, 2004. Fast discovery of connection subgraphs. In: Proceedings of KDD. ACM, pp. 118–127.

Fischetti, Matteo, Leitner, Markus, Ljubić, Ivana, Luipersbeck, Martin, Monaci, Michele, Resch, Max, Salvagnin, Domenico, Sinnl, Markus, 2017. Thinning out Steiner trees: a node-based model for uniform edge costs. Math. Program. Comput. 9 (2), 203–229.

Fischetti, Matteo, Lodi, Andrea, 2003. Local branching. Math. Program. 98 (1), 23–47.

Fischetti, Matteo, Monaci, Michele, 2014. Proximity search for 0-1 mixed-integer convex programming. J. Heuristics 20 (6), 709–731.

Fu, Luoyi, Zhang, Jiapeng, Wang, Shuaiqi, Wu, Xinyu, Wang, Xinbing, Chen, Guihai, 2020. De-anonymizing social networks with overlapping community structure. IEEE/ACM Trans. Netw. 28 (1), 360–375.

Goldberg, Andrew V., Tarjan, Robert E., 1988. A new approach to the maximum-flow problem. J. ACM 35 (4), 921–940.

Gouveia, Luis, Telhada, João, 2008. The multi-weighted Steiner tree problem: A reformulation by intersection. Comput. Oper. Res. 35 (11), 3599–3611.

Held, Stephan, Korte, Bernhard, Rautenbach, Dieter, Vygen, Jens, 2011. Combinatorial optimization in VLSI design. In: Combinatorial Optimization. IOS Press, pp. 33–96.

Klein, Philip, Ravi, R.J.J.A., 1995. A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms 19 (1), 104–115.

Lai, Katherine J., Gomes, Carla P., Schwartz, Michael K., McKelvey, Kevin S., Calkin, David E., Montgomery, Claire A., 2011. The Steiner multigraph problem: wildlife corridor design for multiple species. In: Proceedings of AAAI. AAAI, pp. 1357–1364.

Lappas, Theodoros, Liu, Kun, Terzi, Evimaria, 2009. Finding a team of experts in social networks. In: Proceedings of KDD. ACM, pp. 467–476.

Leggieri, Valeria, Haouari, Mohamed, Triki, Chefi, 2012. A branch-and-cut algorithm for the Steiner tree problem with delays. Optim. Lett. 6 (8), 1753–1771.

Ljubić, Ivana, 2021. Solving Steiner trees: Recent advances, challenges, and perspectives. Networks 77 (2), 177–204.

Ljubic, Ivana, Weiskircher, René, Pferschy, Ulrich, Klau, Gunnar W., Mutzel, Petra, Fischetti, Matteo, 2005. Solving the prize-collecting Steiner tree problem to optimality. ALENEX/ANALCO 2005, 68–76.

Pašić, Alija, Girão-Silva, Rita, Mogyorósi, Ferenc, Vass, Balázs, Gomes, Teresa, Babarczi, Péter, Revisnyei, Péter, Tapolcai, János, Rak, Jacek, 2021. eFRADIR: An enhanced framework for disaster resilience. IEEE Access 9, 13125–13148.

Salhieh, Ayad, Weinmann, Jennifer, Kochhal, Manish, Schwiebert, Loren, 2001. Power efficient topologies for wireless sensor networks. In: Proceedings of ICPP. IEEE, pp. 156–163.

Taccari, Leonardo, 2016. Integer programming formulations for the elementary shortest path problem. European J. Oper. Res. 252 (1), 122–130.

Tarjan, Robert, 1972. Depth-first search and linear graph algorithms. SIAM J. Comput. 1 (2), 146–160.

West, Douglas Brent, et al., 2001. Introduction to Graph Theory, Vol. 2. Prentice hall Upper Saddle River.

Wolsey, Laurence A., 2020. Integer Programming 2nd Edition. John Wiley & Sons.

Zhang, Jianan, Modiano, Eytan, Hay, David, 2017. Enhancing network robustness via shielding. IEEE/ACM Trans. Netw. 25 (4), 2209–2222.