Soft Computing
https://doi.org/10.1007/s00500-023-09541-7

OPTIMIZATION

Optimization algorithms for the *k* edge-connected *L*-hop-constrained network design problem

I. Diarrassouba[1] · A. R. Mahjoub[2,3] · I. M. Almudahka[2]

Accepted: 22 November 2023

Abstract
In this paper, we study the k edge-connected L-hop-constrained network design problem. Given a weighted graph $G = (V, E)$, a set D of pairs of nodes, two integers $L \geq 2$ and $k \geq 2$, the problem consists in finding a minimum weight subgraph of G containing at least k edge-disjoint paths of length at most L between every pair $\{s, t\}$ of D. The problem has several applications in telecommunications network design. It also has applications in reliable container transportation network design and public transportation. Even if the problem has been studied for several decades, it appears, to the best of our knowledge, that the associated polytope is not well known, even when $L \in \{2, 3\}$. In this paper, we are particularly interested in the polyhedral analysis of the problem as well as an exact solving of the problem for large-scale instances. We consider the case where $L \in \{2, 3\}$ and present two integer programming formulations introduced in Diarrassouba et al. (2016). Then, we consider the polytope associated with these formulations, and present several new classes of valid inequalities, involving the so-called design variables. We also present separation algorithms for these inequalities and devise Branch-and-Cut algorithms for solving the problem. Finally, we present an extensive computational study for $L \in \{2, 3\}$ and $k \in \{3, 4, 5\}$, in which we test the efficiency of our Branch-and-Cut algorithms. We also compare the algorithm against a resolution using CPLEX Branch-and-Cut and CPLEX Benders Decomposition frameworks. The results show that our Branch-and-Cut algorithm can be competitive against these two frameworks.

Keywords Hop-constrained survivable network · Edge-disjoint paths · Hop-constrained path · Valid inequalities · Branch-and-Cut algorithm

1 Introduction

Let $G = (V, E)$ be an undirected graph with node set V and edge set E and $D \subseteq V \times V$ a set of pairs of nodes, called *demands* with $|D| = d$. If a pair $\{s, t\}$ is a demand in D, we call s and t *demand nodes* or *terminal nodes*. Let $L \geq 2$ be a fixed integer. If s and t are two nodes of V, an L-st-path in G is a path between s and t of length at most L, where the length is the number of edges (also called *hops*).

Given a weight function $c : E \rightarrow \mathbb{R}$, which associates the weight $c(e)$ to each edge $e \in E$ and an integer $k \geq 2$, the *k-edge-connected L-hop-constrained network design problem* (kHNDP for short) consists in finding a minimum cost subgraph of G having at least k edge-disjoint L-st-paths between each demand $\{s, t\} \in D$. In the remainder of the paper, we will denote by S_D (resp. T_D) the set of origin (resp. destinations) nodes of the demands.

The kHNDP has applications in the design of survivable telecommunication networks where bounded-length paths are required. Survivable networks must satisfy some connectivity requirements; that is, the networks should still be functional after the failure of certain links. As pointed out in Ko and Monma (1989), the topology that seems to be very efficient (and needed in practice) is the uniform topology,

✉ A. R. Mahjoub
ali.mahjoub@ku.edu.kw;
ridha.mahjoub@lamsade.dauphine.fr

I. Diarrassouba
diarrasi@univ-lehavre.fr

I. M. Almudahka
intesar.m@ku.edu.kw

[1] Normandie Univ, UNIHAVRE, LMAH, FR-CNRS-3335, 76600 Le Havre, France

[2] Department of Statistics and Operations Research, College of Science, Kuwait University, Kuwait City, Kuwait

[3] Laboratoire LAMSADE, CNRS UMR 7243, Université Paris-Dauphine, PSL, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France

Published online: 05 February 2024

that is to say that corresponding to networks that survive after the failure of $k-1$ or fewer edges, for some $k \geq 2$. However, this requirement is often insufficient regarding the reliability of a telecommunication network. In fact, the alternative paths could be too long to guarantee an effective routing. In data networks, such as the Internet, the elongation of the route of the information could cause a major loss in the transfer speed. For other networks, the signal itself could be degraded by a longer routing. In such cases, the L-path requirement guarantees exactly the needed quality of the alternative routes. Moreover, in a telecommunication network, usually several commodities have to be routed in the network between pairs of terminals. In order to ensure an effective routing, there must exist a sufficient number of hop-constrained paths between each pair of terminals.

One can also find application of the kHNDP when designing reliable a container shipping service in martitime transportation. Indeed, one of the key issue for container liner shipping companies is to limit as much as possible the number of transshipments for each container, and this, for reducing the chance of losing the container (see Balakrishnana and Vad Karsten (2017)). Moreover, ensuring edge- or node-disjoint maritime routes guarantees that the liner shipping network is less vulnerable to disruption (see Lhomme (2016)).

The hop constraint is frequently met in public transportation systems. In fact, suppose that we want to design a transportation network that uses different transportation means like buses, metro and tram, in order to interconnect a set of origin–destinations (O_i, D_i), $i = 1, ..., n$, in such a way that any passenger can go from an origin O_i to a destination D_i by making at most L changes from one vehicle to another (usually $L = 2$ or 3). For reliability reasons, we may also impose that if one or more of the components of the network (a link or a node) becomes obsolete, the passenger can always reach his destination (by making at most L changes). If we associate with each link a construction cost and we would design a minimum cost network satisfying these conditions, the network design problem here is nothing but the kHNDP.

The kHNDP has been extensively investigated when there is only one demand in the network ($|D| = 1$). In particular, the associated polytope has received special attention. Huygens et al. (2004) study the kHNDP for $k = 2$ and $L = 2, 3$. They give an integer programming formulation for the problem and show that the linear programming relaxation of this formulation completely describes the associated polytope. From this, they obtain a minimal linear description of that polytope. They also show that this formulation is no longer valid when $L \geq 4$. Dahl et al. (2006) study the kHNDP when $L = 2$ and $k \geq 2$. They give a complete description of the associated polytope in this case and show that it can be solved in polynomial time using linear programming. Bendali et al. (2010) generalize the results of Huygens et al. (2004) and Dahl et al. (2006) and give a complete description of the polytope of the kHNDP, with $|D| = 1$, for any $k \geq 2$ and $L \in \{2, 3\}$.

Dahl (1999b) considers the kHNDP for $k = 1$ and $L = 3$. He gives a complete description of the dominant of the associated polytope. Dahl and Gouveia (2004) consider the directed hop-constrained path problem. They describe valid inequalities and characterize the associated polytope when $L \leq 3$. Huygens and Mahjoub (2007) study the kHNDP when $k = 2$ and $L \geq 4$. They also study the variant of the problem where k node-disjoint paths of length at most L are required between two terminals. They give an integer programming formulation for these two problems when $L = 4$.

Diarrassouba et al. (2016) consider the problem when $L \in \{2, 3\}$ and $k \geq 2$. They introduce four extended integer programming formulations and compare these formulations to the so-called natural formulation. They show that for $L \in \{2, 3\}$, the LP-bound provided by these extended formulations has the same value as that provided by the natural formulation. They also solve, for $L = 2, 3$ and $k = 3, 4, 5$, some instances of the problem using CPLEX and compare the formulations in terms of efficiency.

Mahjoub et al. (2013) propose an extended formulation for the kHNDP when L belongs to 2, 3, $k \geq 2$, and the demands are rooted, called hop-level multicommodity flow. They show that this formulation is quite efficient, it permitted to solve several instances from the for the first time. Mahjoub et al. (2019) generalize the technique in Mahjoub et al. (2013), and propose a new generic way to construct extended formulations for a large class of network design problems with given connectivity requirements. They show that significant gap reductions compared with state-of-the art formulations can be obtained.

Botton et al. (2013) propose an extended formulation for the kHNDP for every $L \geq 2$. They also develop an exact algorithm based on a Benders decomposition method, and report computational results for $L \in \{3, 4, 5\}$, $k \in \{1, 2, 3\}$, and with graphs having up to 21 nodes and $|D| \in \{5, 10\}$.

More recently, Arslan et al. (2020) investigate a variant of the kHNDP in which the designed network is such that there exists an L-st-path after the removal of at most $k-1$ edges from the network, for any $k \geq 2$. They present an integer programming formulation relying on the design variables and present some computational results for this problem.

Coullard et al. (1994) investigate the structure of the polyhedron associated with the st-walks of length L of a graph, where a walk is a path that may go through the same node more than once. They present an extended formulation of the problem, and, using projection, they give a linear description of the associated polyhedron. They also discuss classes of facets of that polyhedron.

The kHNDP has also been studied when $|D| \geq 2$. Dahl and Johannessen (2004) consider the case where $k = 1$ and $L = 2$. They introduce valid inequalities and develop a Branch-and-Cut algorithm. The problem of finding a minimum cost spanning tree with hop-constraints is also considered in Gouveia (1999) and Gouveia and Requejo (2001). Here, the hop-constraints limit to a positive integer H the number of links between the root and any terminal in the network. Dahl (1999a) studies the problem when $H = 2$ from a polyhedral point of view, and gives a complete description of the associated polytope when the graph is a wheel. Finally, Huygens et al. (2007) consider the problem of finding a minimum cost subgraph with at least two edge-disjoint L-hop-constrained paths between each given pair of terminal nodes. They give an integer programming formulation of that problem for $L = 2, 3$ and present several classes of valid inequalities. They also devise separation routines. Using these, they propose a Branch-and-Cut algorithm and discuss some computational results.

Besides hop-constraints, another reliability condition, which is used in order to limit the length of the routing, requires that each link of the network belongs to a ring (cycle) of bounded length. Fortz et al. (2000) consider the 2-node connected subgraph problem with bounded rings. This problem consists in finding a minimum cost 2-node connected subgraph (V, F) such that each edge of F belongs to a cycle of length at most L. They describe several classes of facet-defining inequalities for the associated polytope and devise a Branch-and-Cut algorithm for the problem. Fortz et al. (2006) study the edge version of that problem. They give an integer programming formulation for the problem in the space of the natural design variables and describe different classes of valid inequalities. They study the separation problem for these inequalities and discuss Branch-and-Cut algorithms.

In this work, we are mainly interested in the polyhedral description of the kHNDP polytope when $L \in \{2, 3\}$. We first present the integer programming formulations for the kHNDP introduced by Diarrassouba et al. (2016) when $L = 2, 3$ and $k \geq 2$. We then introduce several classes of valid inequalities for the polytope associated with the so-called natural formulation as well as valid inequalities associated with some extended formulations. Then, we discuss the separation problem associated with these inequalities and devise Branch-and-Cut algorithms for solving the kHNDP. Finally, we present an extensive computational study whose aims first at testing the efficiency of our Branch-and-Cut algorithms for solving the problem. It also aims at comparing our algorithms with other resolution methods, reported in the literature, namely CPLEX Branch-and-Cut and CPLEX Automatic Benders decomposition frameworks.

Notice that in this work, we consider two types of demand sets: rooted demands and disjoint demands. A set of rooted demands is composed of demands which have the same node as source node, while a set of disjoint demands is composed of demands in which each source node is associated with only one destination node and vice-versa.

The paper is organized as follows. In Sect. 2, we present the integer programming formulations introduced by Diarrassouba et al. (2016), and in Sects. 3 and 4, we present new valid inequalities for the problem. Sections 5 and 6 are dedicated to the presentation of our Branch-and-Cut algorithms and the computational study we have conducted. Finally, Sect. 7 gives conclusion.

The rest of this section is devoted to more definitions and notation. An edge $e \in E$ with endnodes u and v is denoted by uv. Given two node subsets W and W', we denote by $[W, W']$ the set of edges having one endnode in W and the other in W'. If $W = \{u\}$, we then write $[u, W']$ for $[\{u\}, W']$. We also denote by $\overline{W}$ the node set $V \setminus W$. The set of edges having only one node in W is called a *cut* and denoted by $\delta(W)$. We will write $\delta(u)$ for $\delta(\{u\})$. Given two nodes $s, t \in V$, a cut $\delta(W)$ such that $s \in W$ and $t \in \overline{W}$ is called an *st-cut*.

We will also denote by $H = (U, A)$ a directed graph where U is the set of nodes and A is the set of arcs. An arc a with origin u and destination v will be denoted by (u, v). Given two node subsets W and W' of U, we will denote by $[W, W']$ the set of arcs whose origin is in W and destination in W'. As before, we will write $[u, W']$ for $[\{u\}, W']$ and $\overline{W}$ will denote the node set $U \setminus W$. The set of arcs having their origin in W and their destination in $\overline{W}$ is called a *cut* or *dicut* in H and is denoted by $\delta^+(W)$. We will also write $\delta^+(u)$ for $\delta^+(\{u\})$ with $u \in U$. If s and t are two nodes of H such that $s \in W$ and $t \in \overline{W}$, then $\delta^+(W)$ will be called an *st-cut* or *st-dicut* in H. If W and W' are two node subsets of H, then $[W, W']^+$ will denote the set of arcs of H whose origins are in W and destinations in W'. As for undirected graphs, we will write $[u, W']^+$ for $[\{u\}, W']^+$.

Given an undirected graph $G = (V, E)$ (resp. a directed graph $H = (U, A)$) and an edge subset $F \subseteq E$ (resp. an arc subset $B \subseteq A$), we let $x^F \in \mathbb{R}^E$ (resp. $y^B \in \mathbb{R}^A$) be the *incidence vector* of F (resp. B), that is the $0-1$ vector such that $x^F(e) = 1$ if $e \in F$ (resp. $y^B(a) = 1$ if $a \in B$) and 0 otherwise. Given F a subset of E (resp. A) and a vector $x \in \mathbb{R}^E$ (resp. $y \in \mathbb{R}^A$), $x(F)$ (resp. $y(F)$) will represent the term $\sum_{e \in F} x(e)$ (resp. $\sum_{e \in F} y(e)$). If $\pi = (V_1, ..., V_p)$, $p \geq 2$, is a partition of V, then we denote by $\delta(\pi)$ the set of edges having their endnodes in different sets. Given a partition $\pi = (V_1, ..., V_p)$, $p \geq 2$, we will denote by G_π the subgraph induced by π that is the graph obtained from G by contracting the sets V_i, for $i = 1, ..., p$. Note that the edge set of G_π is the set $\delta(V_1, ..., V_p)$.

2 Integer programming formulations

In this section, we present two integer programming formulations, presented by Diarrassouba et al. (2016), for the kHNDP when $L = 2, 3$. The first formulation is the so-called natural formulation whose variables set corresponds to the set of edges of the input graph while the second formulation is an extended formulation based on auxiliary graphs.

2.1 The natural formulation

Let $G = (V, E)$ be an undirected graph, $D \subseteq V \times V$ be a demand set, and two integers $k \geq 2$ and $L \in \{2, 3\}$. If an edge subset $F \subseteq E$ induces a solution of the kHNDP, that is a subgraph (V, F) contains k-edge-disjoint L-st-paths for every $\{s, t\} \in D$, then its incidence vector x satisfies the following inequalities:

$$x(\delta(W)) \geq k \text{ for all } st\text{-cuts } \delta(W),\ W \subset V, \{s, t\} \in D, \tag{2.1}$$

$$x(e) \geq 0 \qquad \text{for all } e \in E, \tag{2.2}$$

$$x(e) \leq 1 \qquad \text{for all } e \in E. \tag{2.3}$$

Inequalities (2.1) are the so-called *st-cut inequalities* while (2.2) and (2.3) are the *trivial inequalities*.

Now, for a demand $\{s, t\} \in D$, we consider the partition $\pi = (V_0, V_1, ..., V_{L+1})$ of V such that $s \in V_0$, $t \in V_{L+1}$ and $V_i \neq \emptyset$, for all $i \in \{1, ..., L\}$. Let T be the set of edges $e = uv$, where $u \in V_i$, $v \in V_j$, and $|i - j| > 1$. The edge set T is called an *L-st-path-cut*. Figure 1 gives an example of L-st-path-cut with $L = 3$ and, $V_0 = \{s\}$ and $V_{L+1} = \{t\}$.

The following inequality:

$$x(T) \geq k \tag{2.4}$$

is the so-called *L-st-path-cut inequality* induced by π. Dahl (1999b) showed that the L-st-path-cut (2.4) inequalities are valid for the kHNDP polytope when $k = 1$ and $|D| = 1$. It is not hard to see that L-st-path-cut inequalities (2.4) are valid for the kHNDP polytope for any $k \geq 1$, $L \geq 2$ and $|D| \geq 2$.

It can be shown (see Diarrassouba et al. (2016) and Huygens et al. (2007)) that L-st-path-cut inequalities together with the st-cut inequalities (2.1), (2.2) and (2.3) provide an integer programming formulation for the kHNDP when $L = 2, 3$.

Theorem 2.1 Huygens et al. (2007) *Let $G = (V, E)$ be a graph, $k \geq 2$ and $L \in \{2, 3\}$. Then, the kHNDP is equivalent to the following integer program:*

$$min\{cx;\ \textit{subject to } (2.1) - -(2.4),\ x \in \mathbb{Z}^E\}. \tag{2.5}$$

Formulation (2.5) is called *the natural formulation* and is denoted by $k\text{HNDP}_{Nat}$. The associated polytope is denoted by $k\text{HNDP}_{Nat}(G, D)$, that is

$$k\text{HNDP}_{Nat}(G, D) = \{x \in \mathbb{Z}^E \text{ whichsatisfies} (2.1) - (2.4)\}.$$

Huygens et al. (2007) study the polytope associated with this formulation and introduce some facet-defining inequalities for the problem. They also develop a Branch-and-Cut algorithm for the kHNDP when $k = 2$ and $L = 2, 3$.

Bendali et al. (2010) show that an L-st-path-cut inequality induced by a partition $(V_0, ..., V_{L+1})$, with $s \in V_0$ and $t \in V_{L+1}$, is facet-defining if only if $|V_0| = |V_{L+1}| = 1$. Therefore, in the remainder of the paper, the only L-st-path-cuts that we will consider are those induced by partitions of the form $(\{s\}, V_1, ..., V_L, \{t\})$.

2.2 A separated flow formulation

In this section, we present three integer programming formulations for the kHNDP for $L = 2, 3$ where we use a directed layered graph to model each hop-constrained subproblem. These formulations are called *separated formulations*. The graph transformation supporting these formulations is given below.

Given $G = (V, E)$ and $\{s, t\} \in D$, let $\widetilde{G}_{st} = (\widetilde{V}_{st}, \widetilde{A}_{st})$ be the layered digraph obtained from G as follows:

- $\widetilde{V}_{st} = N_{st} \cup N'_{st} \cup \{s, t\}$ with $N_{st} = V \backslash \{s, t\}$ and N'_{st} is a copy of N_{st} (each node $u \in N_{st}$ corresponds to a node u' of N'_{st}),
- $\widetilde{A}_{st}$ is composed of four kinds of arcs:
  - for all $su \in E$, $(s, u) \in \widetilde{A}_{st}$,
  - for all $vt \in E$, $(v', t) \in \widetilde{A}_{st}$,
  - for all $u \in N_{st}$, we introduce $\min\{|[s, u]|, |[u, t]|, k\}$ arcs of the form $(u, u') \in \widetilde{A}_{st}$,
  - if $L = 3$, for all $uv \in E \backslash \{st\}$ with $u, v \in N_{st}$, $\{(u, v'), (v, u')\} \in \widetilde{A}_{st}$ (see Fig. 2 for an illustration with $L = 3$).

For an edge $e = uv \in E$, we denote by $\widetilde{A}_{st}(e)$ the set of arcs of $\widetilde{G}_{st}$ corresponding to the edge e:

- when $u = s$ (resp. $v = t$), $\widetilde{A}_{st}(e)$ contains (s, v) (resp. (u', t)),
- when $u \neq s$ and $v \neq t$, if $L = 3$, $\widetilde{A}_{st}(e) = \{(u, v'), (v, u')\}$ and, if $L = 2$, $\widetilde{A}_{st}(e)$ is empty.

Note that $\widetilde{G}_{st}$ may have nodes different from $u \in N_{st} \cup N'_{st}$ with indegree or outdegree equal to zero. These nodes can be removed from $\widetilde{G}_{st}$ after its construction.

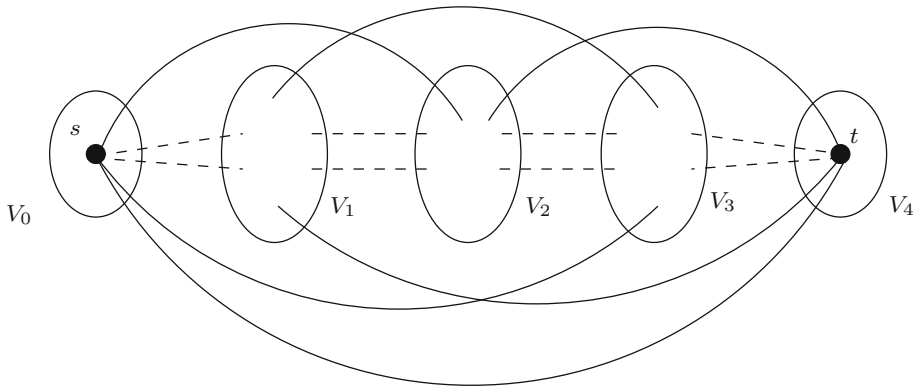

Fig. 1 Support graph of a L-st-path-cut with $L = 3$, $V_0 = \{s\}$, $V_{L+1} = \{t\}$ and T formed by the solid edges

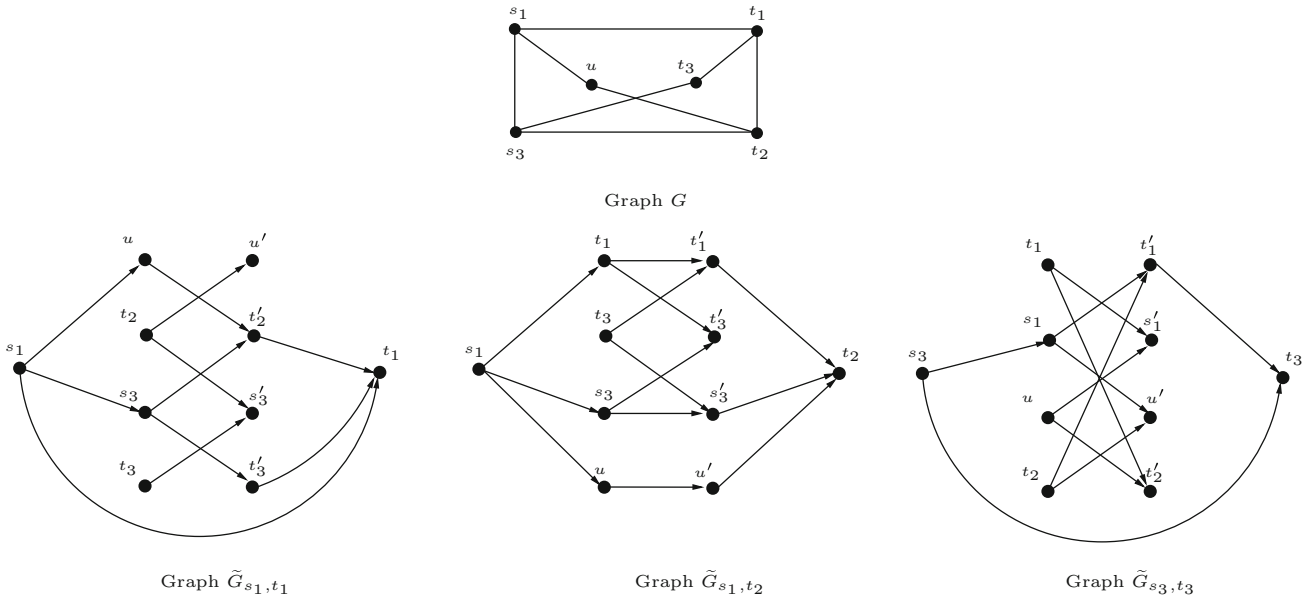

Fig. 2 Construction of graphs $\widetilde{G}_{st}$ with $D = \{\{s_1, t_1\}, \{s_1, t_2\}, \{s_3, t_3\}\}$ for $L = 3$ and $k = 1$

$\widetilde{G}_{st}$ contains four layers: $\{s\}$, N_{st}, N'_{st}, $\{t\}$ and no circuit. Also, any st-dipath in $\widetilde{G}_{st}$ is of length no more than 3:

- the length is equal to 1 if the st-dipath is composed of the single arc (s, t),
- the length is equal to 3 for both st-dipaths of the form (s, u, u', t) corresponding to path (s, u, t) of length 2 in G, and st-dipaths of the form (s, u, v', t) corresponding to path (s, u, v, t), with $u \neq v$, of length 3 in G.

Diarrassouba et al. (2016) (see also Bendali et al. (2010)) pointed out that a set of k arc-disjoint st-dipaths of $\widetilde{G}_{st}$ corresponds to a set of k edge-disjoint 3-st-paths in G, and vice-versa. Thus, a solution of kHNDP can be obtained by computing st-flows in $\widetilde{G}_{st}$ with appropriate capacities on the arcs, together with some additional constraints. Such a formulation, called *Separated Flow Formulation*, is described below.

Given a demand $\{s, t\}$, we let $f^{st} \in \mathbb{R}^{\widetilde{A}_{st}}$ be a flow vector on $\widetilde{G}_{st}$ of value k between s and t.

Then, f^{st} satisfies the *flow conservation constraints* (2.6), given by

$$\sum_{a\in\delta^+(u)} f_a^{st} - \sum_{a\in\delta^-(u)} f_a^{st} = \left\{ \begin{array}{r} k \text{ if } u = s, \\ 0 \text{ if } u \in \widetilde{V}_{st} \setminus \{s, t\}, \\ -k \text{ if } u = t, \end{array} \right\}, \quad \text{for all } u \in \widetilde{V}_{st}, \{s, t\} \in D \tag{2.6}$$

and

$$f_a^{st} \leq x(e), \text{ for all } a \in \widetilde{A}_{st}(e), e \in E, \{s, t\} \in D. \tag{2.7}$$

$$f_a^{st} \leq 1, \text{ for all } a = (u, u'), \ u \in V \setminus \{s, t\}, \ \{s, t\} \in D. \tag{2.8}$$

$$f_a^{st} \geq 0, \text{ for all } a \in \widetilde{A}_{st}, \ \{s, t\} \in D. \tag{2.9}$$

$$x(e) \leq 1, \text{ for all } e \in E. \tag{2.10}$$

Inequalities (2.7) are also called *linking inequalities*. They indicate that if an edge $e \in E$ is not in the solution, then the flow on every arc corresponding to e is 0. Inequalities (2.9)-(2.10) are the *trivial inequalities*.

Thus, we have the following theorem.

Theorem 2.2 Diarrassouba et al. (2016) *The kHNDP for $L = 2, 3$ is equivalent to the following integer program:*

$$min\{cx;\ subject\ to\ (2.6)-(2.10),\ x \in \mathbb{Z}_+^E,\ f^{st} \in \mathbb{R}_+^{\widetilde{A}_{st}},\ for\ all\ \{s,t\} \in D\}. \tag{2.11}$$

Formulation (2.11) will be called *the separated flow formulation* and will be denoted by $k\text{HNDP}_{Flow}^{Sep}$.

In the following sections, we present several classes of valid inequalities for the kHNDP polytopes. These inequalities rely on the design variables, that are variables $x(e)$. Thus, they are valid for the polytope induced by both the Natural and Separated Flow Formulations.

We start with the so-called partition inequalities for $k \geq 2$ and $L = 2, 3$. To the best of our knowledge, these inequalities have never been presented before in the literature.

3 Hop-constrained partition inequalities for $k\text{HNDP}_{Nat}(G, D)$

Our new inequalities are defined for any $k \geq 2$ and for $L \in \{2, 3\}$. Also we distinguish the case where the demands are rooted or disjoint. We recall that S_D and T_D denote the set of origins and the set of destinations of the demands, respectively. When the demands are rooted, the set S_D is reduced to one node s, that is $S_D = \{s\}$, while for disjoint demands, each source node is associated with only one destination node and vice-versa.

The main results of the present section are given in Theorems 3.1 and 3.2.

3.1 Hop-constrained partition inequalities for rooted demands

We first consider the case of rooted demands, that is $S_D = \{s\}$. Let $\pi = \{V_0, V_1, ..., V_p\}$, $p \geq 2$, be a partition of V such that $s \in V_0$ and $V_i \cap T_D \neq \emptyset, i = 1, ..., p$. The inequalities

$$x(\delta(V_0, V_1, ..., V_p)) \geq \begin{cases} \left\lceil \frac{(k+1)p}{2} \right\rceil, & \text{if } L = 2, \quad (3.1) \\ \left\lceil \frac{\left(k + \frac{2}{k+1}\right)p}{2} \right\rceil, & \text{if } L = 3, \end{cases} \tag{3.2}$$

are called *Hop-Constrained Partition inequalities* (HC-Partition inequalities for short). The following theorem states that these inequalities are valid for the kHNDP when the demands are rooted. For the proof, see Diarrassouba and Mahjoub (2023).

Theorem 3.1 *Consider the kHNDP with rooted demands and let $\pi = \{V_0, V_1, ..., V_p\}$, $p \geq 2$, be a partition of V such that $s \in V_0$ and $V_i \cap T_D \neq \emptyset$, $i = 1, ..., p$. The partition inequality (3.1) (resp. (3.2)), induced by π, is valid for the kHNDP for any $k \geq 2$ and when $L = 2$ (resp. $L = 3$).*

Figures 3 and 4 present examples of partition supporting partition inequalities in cases $L = 2$ and $L = 3$, respectively.

Figures 3 and 4 present partitions with $p = 6$ and $p = 15$, respectively, and where $|V_i| = 1$, for every $i \in \{0, ..., p\}$. We can see that the solution depicted in Fig. 3 is feasible for the kHNDP for $L = 2$ and $k = 3$. Also, the number of edges in $\delta(\pi)$ is $12 = \left\lceil \frac{(3+1)\cdot 6}{2} \right\rceil$. For Fig. 4, the solution is feasible for the kHNDP with $L = 3$ and $k = 4$, and the number of edges in $\delta(\pi)$ is $33 = \left\lceil \frac{(4+2/5)\cdot 15}{2} \right\rceil$.

Next, we address the case where the demands are disjoint.

3.2 Hop-constrained partition inequalities for disjoint demands

Consider the kHNDP with disjoint demands and let $\pi = \{V_0, V_1, ..., V_p\}$, $p \geq 2$, be a partition of V such that $V_0 \supseteq S_D$ and $V_i \cap T_D \neq \emptyset, i = 1, ..., p$. The inequalities

$$x(\delta(V_0, V_1, ..., V_p)) \geq \begin{cases} \left\lceil \frac{\left(k + \lceil \frac{k}{2} \rceil\right)p}{2} \right\rceil, & \text{if } L = 2, \quad (3.3) \\ \left\lceil \frac{\left(k + \frac{k}{k+1}\right)p}{2} \right\rceil, & \text{if } L = 3, \end{cases} \tag{3.4}$$

are called *Hop-Constrained partition inequalities*.

Theorem 3.2 *Consider the kHNDP with disjoint demands and let $\pi = \{V_0, V_1, ..., V_p\}$, $p \geq 2$, be a partition such that $V_0 \supseteq S_D$ and $V_i \cap T_D \neq \emptyset$, $i = 1, ..., p$. The partition inequality (3.3) (resp. (3.4)), induced by π, is valid for the kHNDP for any $k \geq 2$ and when $L = 2$ (resp. $L = 3$).*

The proof of Theorem 3.2 is given in Diarrassouba and Mahjoub (2023).

Figures 5 and 6 illustrate the support graphs of partition inequalities in cases $L = 2$ and $L = 3$, respectively, and disjoint demands.

In Fig. 5, the demand set is composed of pairs (s_i, t_i), $i = 1, ..., 4$, and the considered partition is $\pi = \{V_0, V_1, ..., V_p\}$,

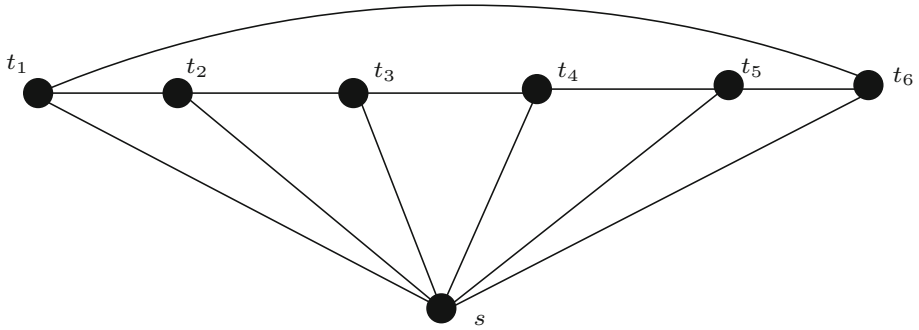

Fig. 3 Support graph of a partition inequality for rooted demands, $L = 2$ and $k = 3$, $p = 6$, $V_0 = \{s\}$ and $V_i = \{t_i\}$, $i = 1, ..., p$

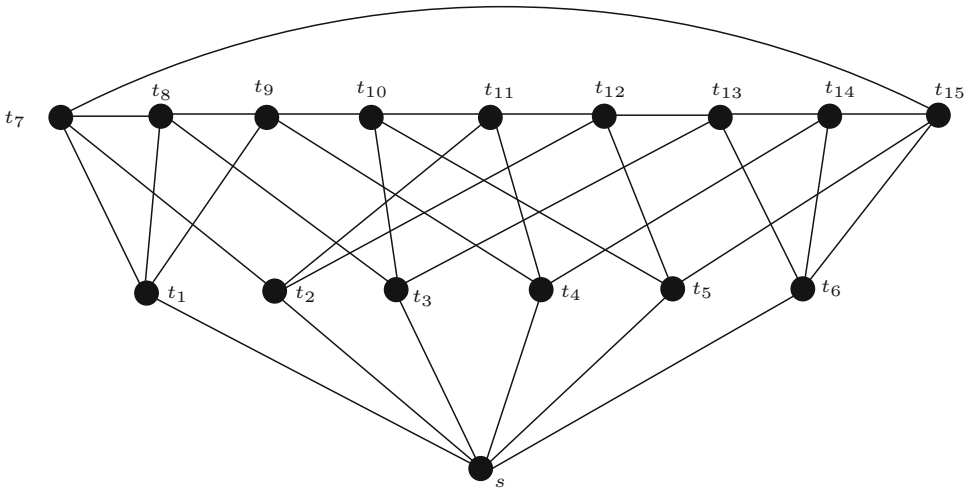

Fig. 4 Support graph of a partition inequality for rooted demands, $L = 3$ and $k = 4$, $p = 15$, $V_0 = \{s\}$ and $V_i = \{t_i\}$, $i = 1, ..., p$

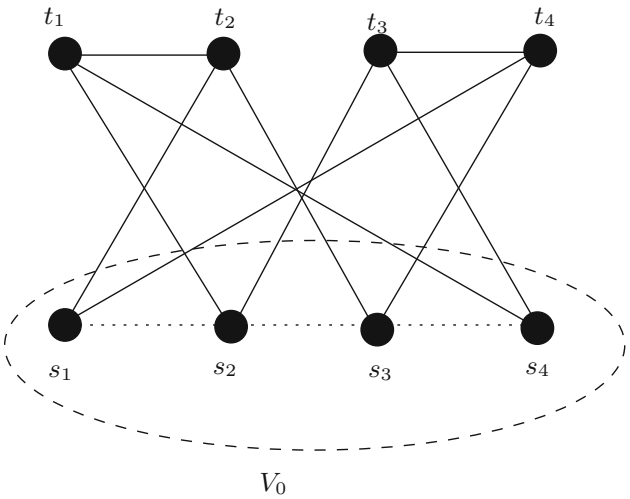

Fig. 5 Support graph of a partition inequality for disjoint demands, $L = 2$ and $k = 3$. Here, $p = 4$, $V_i = \{t_i\}$, $i = 1, ..., 4$, and $V_0 = \{s_1, ..., s_4\}$

with $p = 4$, and $V_1 = \{t_i\}$, $i = 1, ..., 4$, and $V_0 = \{s_1, s_2, s_3, s_4\}$. It is easy to see that the solution presented here is feasible for the kHNDP with $L = 2$ and $k = 3$. Also, the number of edges involved in $\delta(\pi)$ is $10 = \left\lceil \frac{5 \cdot 4}{2} \right\rceil$.

With Fig. 6, the demand set is composed of pairs (s_i, t_i), $i = 1, ..., 10$. The partition is obtained for $p = 10$ and $V_i = \{t_i\}, i = 1, ..., 10$ and $V_0 = \{s_1, ..., s_{10}\}$. Here also, the solution is feasible for the kHNDP with $k = 4$ and $L = 3$ and the number of edges in $\delta(\pi)$ is $24 = \left\lceil \frac{(4+4/5)\cdot 10}{2} \right\rceil$.

Remark that the solutions presented in Figs. 3, 4, 5 and 6 satisfy with equality the partition inequality associated with each situation. This remark suggests that, under suitable conditions, the partition inequalities (3.1)–(3.4) may define facets of the kHNDP polytope.

4 Further valid inequalities for kHNDP$_{Nat}(G, D)$

In this section, we present three other classes of inequalities which are valid for the kHNDP polytope. These inequalities, with their proofs, are introduced in Diarrassouba (2009).

4.1 Double cut inequalities

In the following, we introduce a class of inequalities that are valid for the kHNDP polytope for $L \geq 2$ and $k \geq 2$. They are given by the following theorem.

Theorem 4.1 *Let $\{s, t\}$ be a demand, $i_0 \in \{0, ..., L\}$ and*

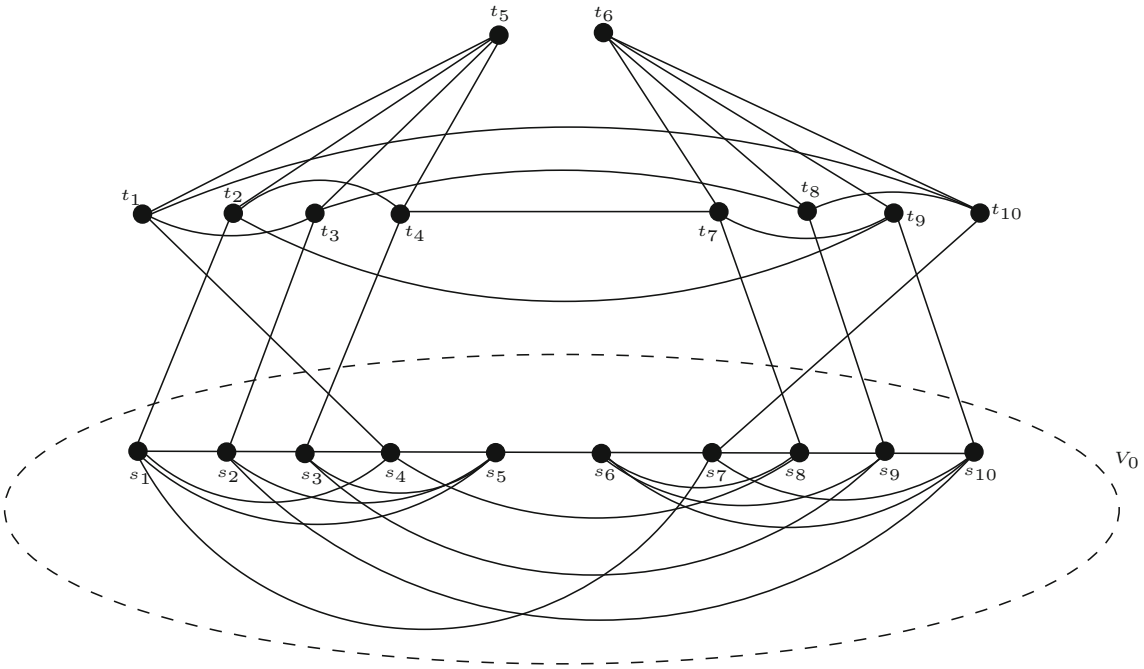

Fig. 6 Support graph of a partition inequality for disjoint demands, $L = 3$ and $k = 4$, $p = 10$, $V_i = \{t_i\}$, $i = 1, ..., p$, and $V_0 = \{s_1, ..., s_p\}$

$\Pi = \{V_0, ..., V_{i_0-1}, V^1_{i_0}, V^2_{i_0}, V_{i_0+1}, ..., V_{L+1}\}$ *a family of node sets of* V *such that*

$\pi = (V_0, ..., V_{i_0-1}, V^1_{i_0}, V^2_{i_0} \cup V_{i_0+1}, V_{i_0+2}, ..., V_{L+1})$ *induces a partition of* V. *Suppose that*

1. $V^1_{i_0} \cup V^2_{i_0}$ *induces an* $s_{j_1}t_{j_1}$*-cut of* G *with* $\{s_{j_1}, t_{j_1}\} \in D$ *and* $s_{j_1} \in V^1_{i_0}$ *or* $t_{j_1} \in V^1_{i_0}$ *(note that* s_{j_1} *and* t_{j_1} *cannot be simultaneously in* $V^1_{i_0}$ *and are not in* $V^2_{i_0}$. *Also note that* $V^2_{i_0}$ *may be empty);*
2. V_{i_0+1} *induces an* $s_{j_2}t_{j_2}$*-cut of* G *with* $\{s_{j_2}, t_{j_2}\} \in D$ *(note that* j_1 *and* j_2 *may be equal);*
3. π *induces an* L*-*st*-path-cut of* G *with* $s \in V_0$ *(resp.* $t \in V_0$*) and* $t \in V_{L+1}$ *(resp.* $s \in V_{L+1}$*).*

Let $\overline{E} = [V_{i_0-1}, V^1_{i_0}] \cup [V_{i_0+2}, V^2_{i_0} \cup V_{i_0+1}] \cup \left(\bigcup_{k,l \notin \{i_0, i_0+1\}, |k-l|>1} [V_k, V_l] \right)$ *and* $F \subseteq \overline{E}$ *such that* $|F|$ *and* k *have different parities.*

Let also $\hat{E} = (\bigcup_{i=0}^{i_0-2} [V_i, V_{i+1}]) \cup (\bigcup_{i=i_0+2}^{L} [V_i, V_{i+1}]) \cup F$. *Then, the inequality*

$$x(\delta(\pi) \setminus \hat{E}) \geq \left\lceil \frac{3k - |F|}{2} \right\rceil, \tag{4.1}$$

is valid for kHNDP$_{Nat}(G, D)$. *(recall that* $\delta(\pi)$ *is the set of edges of the* E *having their endnodes in different elements of* π*).*

Inequalities of type (4.1) are called *double cut inequalities*.

The set of edges having a positive coefficient in inequality (4.1) plus the edges of F is called a *double cut*. Figure 7 gives an example for $L = 3$ and $i_0 = 0$.

Let $L = 2$, $\{s, t\} \in D$ and $\Pi = \{V^1_0, V^2_0, V_1, V_2, V_3\}$ be a family of node sets of V such that $\pi = (V^1_0, V^2_0 \cup V_1, V_2, V_3)$ induces a 2-st-path-cut, and V_1 induces a valid s_1t_1-cut in G, for some $\{s_1, t_1\} \in D$. If $F \subseteq [V^2_0 \cup V_1, V_2]$ is chosen such that $|F|$ and k have different parities, then the double cut inequality induced by Π and F in this case can be written as

$$x([V^1_0, V_1 \cup V_2 \cup V_3]) + x([V^2_0, V_1 \cup V_3]) + x([V_1, V_3]) + x([V^2_0 \cup V_1, V_2] \setminus F) \geq \left\lceil \frac{3k - |F|}{2} \right\rceil. \tag{4.2}$$

Now, let $L = 3$, $\{s, t\} \in D$ and $\Pi = \{V^1_0, V^2_0, V_1, V_2, V_3, V_4\}$ be a family of node sets of V such that $\pi = (V^1_0, V^2_0 \cup V_1, V_2, V_3, V_4)$ induces a 3-st-path-cut, and V_1 induces a valid s_1t_1-cut in G. If $F \subseteq [V^2_0 \cup V_1 \cup V_4, V_2]$ is chosen such that $|F|$ and k have different parities, then the double cut inequality induced by Π and F can be written as

$$x([V^1_0, V_1 \cup V_2 \cup V_3 \cup V_4]) + x([V^2_0, V_1 \cup V_3 \cup V_4]) + x([V_1, V_3 \cup V_4]) + x([V^2_0 \cup V_1 \cup V_4, V_2] \setminus F) \geq \left\lceil \frac{3k - |F|}{2} \right\rceil. \tag{4.3}$$

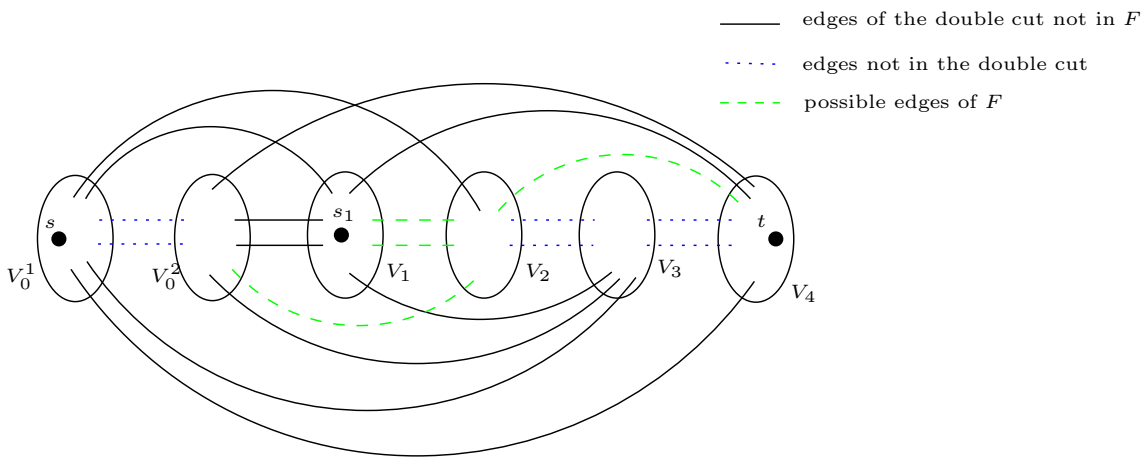

Fig. 7 A double cut with $L = 3$ and $i_0 = 0$

4.2 Triple path-cut inequalities

For this class of valid inequalities, we distinguish the cases where $L = 2$ and $L = 3$.

Theorem 4.2 *i) Let $L = 2$ and $\{V_0, V_1, V_2, V_3^1, V_3^2, V_4^1, V_4^2\}$ be a family of node sets of V such that $(V_0, V_1, V_2, V_3^1 \cup V_3^2, V_4^1 \cup V_4^2)$ induces a partition of V and there exist two demands $\{s_1, t_1\}$ and $\{s_2, t_2\}$ with $s_1, s_2 \in V_0$, $t_1 \in V_3^2$ and $t_2 \in V_4^2$. The sets V_3^1 and V_4^1 may be empty and s_1 and s_2 may be the same. Let also $V_3 = V_3^1 \cup V_3^2$, $V_4 = V_4^1 \cup V_4^2$ and $F \subseteq [V_3^2, V_1 \cup V_4^1] \cup [V_3^1, V_4^2]$ such that $|F|$ and k have different parities. Then, the inequality*

$$2x([V_0, V_2]) + x([V_0, V_3 \cup V_4]) + x([V_4^2, V_1 \cup V_3^2]) + x(([V_3^2, V_1 \cup V_4^1] \cup [V_3^1, V_4^2]) \setminus F) \geq \left\lceil \frac{3k - |F|}{2} \right\rceil \tag{4.4}$$

is valid for kHNDP$_{Nat}(G, D)$.

ii) Let $L = 3$ and $(V_0, ..., V_3, V_4^1, V_4^2, V_5^1, V_5^2)$ be a family of node sets of V such that $(V_0, ..., V_3, V_4^1 \cup V_4^2, V_5^1 \cup V_5^2)$ induces a partition of V and there exist two demands $\{s_1, t_1\}$ and $\{s_2, t_2\}$ with $s_1, s_2 \in V_0$, $t_1 \in V_4^2$ and $t_2 \in V_5^2$. The sets V_4^1 and V_5^1 may be empty and s_1 and s_2 may be the same. Let also $V_4 = V_4^1 \cup V_4^2$, $V_5 = V_5^1 \cup V_5^2$ and $F \subseteq [V_2, V_4^2] \cup [V_3, V_4 \cup V_5]$ such that $|F|$ and k have different parities. Then, the inequality

$$2x([V_0, V_2]) + 2x([V_0, V_3]) + 2x([V_1, V_3]) + x([V_0 \cup V_1, V_4 \cup V_5]) + x([V_4, V_5]) + x([V_2, V_5^2]) + x(([V_2, V_4^2] \cup [V_3, V_4 \cup V_5]) \setminus F) \geq \left\lceil \frac{3k - |F|}{2} \right\rceil \tag{4.5}$$

is valid for kHNDP$_{Nat}(G, D)$.

Inequalities of type (4.4) and (4.5) will be called *triple path-cut inequalities*. The set of edges having a positive coefficient in inequality (4.4) ((4.5)) plus the edges of F will be called a *triple path-cut* (see Fig. 8 for an example with $L = 2$).

4.3 Steiner-SP-partition inequalities

While the previous classes of inequalities combine both edge-connectivity and hop-constraints, the next family models only edge-connectivity. They generalizes the so-called SP-partition inequalities introduced in Chopra (1994) (see also Didi Biha and Mahjoub (1996)) for the k edge-connected subgraph problem. These inequalities are defined as follows.

Let $\pi = (V_1, ..., V_p)$, $p \geq 3$, be a partition of V such that the graph $G_\pi = (V_\pi, E_\pi)$ is series–parallel. Suppose that $V_\pi = \{v_1, ..., v_p\}$ where v_i is the node of G_π corresponding to the set V_i, $i = 1, ..., p$. The partition π is said to be a *Steiner-SP-partition* if and only if π is a Steiner-partition (i.e., every set V_i contains at least one origin or destination node) and either

1. $p = 3$ or
2. $p \geq 4$ and there exists a node $v_{i_0} \in V_\pi$ incident to exactly two nodes v_{i_0-1} and v_{i_0+1} such that the partitions π_1 and π_2 obtained from π by contracting respectively the sets V_{i_0}, V_{i_0-1}, V_{i_0} and V_{i_0+1} are themselves Steiner-SP-partitions.

With a Steiner-SP-partition $(V_1, ..., V_p)$, $p \geq 3$, we associate the following inequality:

$$x(\delta(V_1, ..., V_p)) \geq \left\lceil \frac{k}{2} \right\rceil p - 1. \tag{4.6}$$

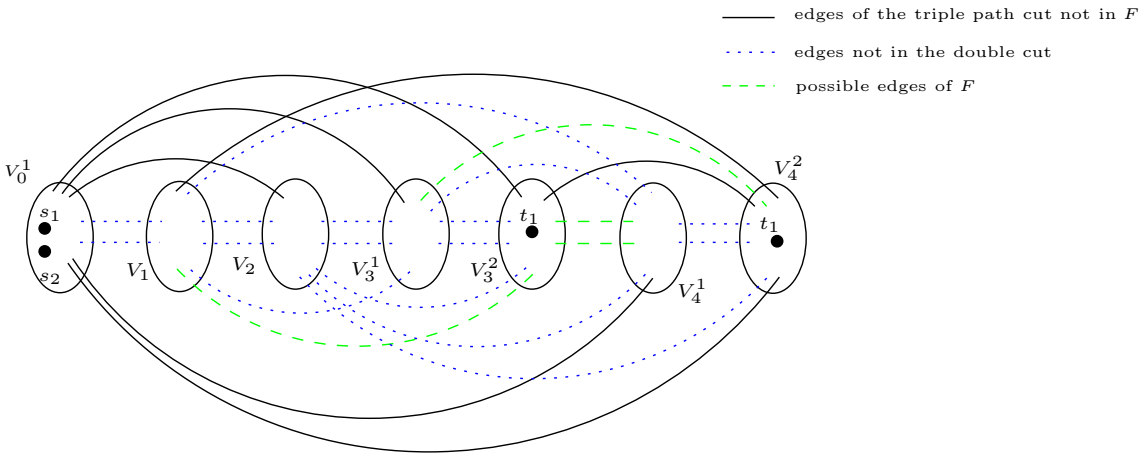

Fig. 8 A triple path-cut with $L = 2$

Inequalities of type (4.6) will be called *Steiner-SP-partition inequalities*. We have the following.

Theorem 4.3 *Inequality (4.6) is valid for* kHNDP$_{Nat}(G, D)$.

Inequality (4.6) expresses the fact that, in a solution of the kHNDP, the multicut induced by a Steiner-SP-partition contains at least $\lceil \frac{k}{2} \rceil p - 1$ edges, since this solution contains k edge-disjoint paths between every pair of nodes $\{s, t\} \in D$.

We also introduce a lifting procedure for Inequalities (4.6) in the case where the graph G_π is not series–parallel. This lifting procedure is similar to that introduced by Chopra (1994) for the kECSP. Let $G = (V, E)$ be a graph and $k \geq 3$ an odd integer. Let $G' = (V, E \cup E')$ be a graph obtained from G by adding an edge set E'. Let $\pi = (V_1, ..., V_p)$ be a Steiner-SP-partition of G. Then, the following inequality is valid for kHNDP$_{Nat}(G, D)$:

$$x(\delta_G(V_1, ..., V_p)) + \sum_{e \in E' \cap \delta_{G'}(V_1, ..., V_p)} a(e)x(e) \geq \left\lceil \frac{k}{2} \right\rceil p - 1, \tag{4.7}$$

where $a(e)$ is the length (in terms of edges) of a shortest path in G_π between the endnodes of e, for all $e \in E' \cap \delta_{G'}(V_1, ..., V_p)$. We will call inequalities of type (4.7) *lifted Steiner-SP-partition inequalities*.

5 Branch-and-Cut algorithms for the kHNDP

In this section, we present the Branch-and-Cut algorithms we have devised to solve the kHNDP. For our purpose, we devise Branch-and-Cut algorithms using the natural formulation and the separated flow formulation. We explain later why we focus only on these algorithms. In Sects. 5.1 and 5.2, we describe the framework of these algorithms, and in Sect. 6, we present some computational results.

Here, we recall some notations and definitions that will be used along this section. Given a solution $x \in [0, 1]^E$, the *support graph* $G(x) = (V, E(x))$ is the subgraph of G obtained by removing from G all the edges $e \in E$ such that $x(e) = 0$, that is $E(x) = \{e \in E \mid x(e) > 0\}$.

In what follows, we briefly describe the Branch-and-Cut algorithms devised for the natural formulation and the separated flow formulation. For each formulation, we present the overall algorithm. Then, we also briefly present the separation algorithm used for the valid inequalities introduced in the paper, as well as a primal heuristic used in both algorithms.

5.1 Branch-and-Cut algorithms for natural and separated flow formulations

To start the optimization, we consider the linear program given by the st-cut inequalities induced by the node sets $\{s\}$ and $\{t\}$, for all $\{s, t\} \in D$, together with the trivial inequalities. That is to say, we consider the program

$$\begin{aligned}
&\text{Min} \sum_{e \in E} c(e)x(e) \\
&\left.\begin{aligned} x(\delta(s)) &\geq k, \\ x(\delta(t)) &\geq k, \end{aligned}\right\} \text{ for all } \{s, t\} \in D, \\
&0 \leq x(e) \leq 1, \text{ for all } e \in E.
\end{aligned}$$

The optimal solution $\overline{x}$ of this LP is feasible for kHNDP$_{Nat}$ if and only if $\overline{x}$ is integral and satisfies every st-cut and L-st-path-cut inequality, for all $\{s, t\} \in D$. If $\overline{x}$ is not feasible for the problem, then we try to generate further valid inequalities for kHNDP$_{Nat}(G, D)$ that are violated by $\overline{x}$. This is done by solving the so-called separation problem. The *separation problem* associated with $\overline{x}$ and a class of inequalities $\mathcal{F}$ consists in saying if there is an inequality of $\mathcal{F}$ which is violated by $\overline{x}$ or not, and if one exists, then exhibit such an inequality. An algorithm for solving this problem is called a *separation*

algorithm. For our Branch-and-Cut algorithm, we solve the separation problem associated with solution of the current LP and the following inequalities, in this order:

1. st-cut and L-st-path-cut inequalities (2.1) and (2.4),
2. Hop-Constrained partition inequalities,
3. Steiner-SP-partition inequalities,
4. Double cut inequalities,
5. Triple path-cut inequalities.

For the separated flow formulation, the optimization starts by solving the linear relaxation of Formulation (2.11). As this formulation contains a polynomial number of variables and constraints, its linear relaxation can be solved using only one linear program:

$$\text{Min} \sum_{e \in E} c(e)x(e)$$

sujected to

(2.6) − (2.10).

The optimal solution $(\overline{x}, \overline{f}^{s_1t_1}, ..., \overline{f}^{s_dt_d})$ of this LP is feasible for $k\text{HNDP}^{Flow}_{Sep}$ if it is integral. If this is not the case, we then try to add further inequalities that are violated by this solution. The inequalities that are considered here are the following and generated in this order:

1. Hop-Constrained partition inequalities,
2. Steiner-SP-partition inequalities,
3. Double cut inequalities,
4. Triple path-cut inequalities.

In the following section, we describe the different separation procedures we use to detect the violated inequalities.

5.2 Separation algorithms

The separation problem of the st-cut and L-st-path-cut inequalities can be solved in polynomial time using network flows in graphs $\widetilde{G}_{st}$, for every $(s,t) \in D$. For the double cut, triple path cut and Steiner-SP-partition inequalities, we use the separation algorithms introduced in Diarrassouba (2009). The three algorithms are heuristics and runs in polynomial time. For the Hop-Constrained Partition inequalities introduced in the present paper, we devise a heuristic which consists in computing a series of flows with lower bounds. The details of the algorithm are given in Diarrassouba and Mahjoub (2023).

5.3 Primal heuristic

An important issue in the efficiency of the Branch-and-Cut algorithms is to compute a good upper bound at each node of the Branch-and-Cut tree. For our Branch-and-Cut algorithms, we use the primal heuristic described in Diarrassouba (2009). This heuristic aims at transforming a fractional solution obtained from the linear relaxation of the problem into a feasible solution for the problem.

6 Computational results

This section is dedicated to the computational results we have obtained with our Branch-and-Cut algorithms, described in the previous section. They have been implemented in C++, using SCIP 7.0.1 (Gamrath et al. 2020) to manage the Branch-and-Bound tree, and CPLEX 12.9 (Cplex 12.9, ibm 2019) as LP-solver. The algorithms have been tested using a machine equipped with a processor Intel Core i7-7820HQ at 2.90 GHz and 32 Go of RAM, running under Linux. The maximum CPU time has been fixed to 5 h (18,000 s).

The tests have been performed for $L = 2, 3$ and $k = 3, 4, 5$. The instances are composed of complete Euclidian graphs from TSPLIB library (Tsplib 1995). The graphs have 21 nodes up to 70 nodes for both $L = 2$ and $L = 3$. The demands used in these instances are randomly generated. Each set of demands is either rooted in a specific node, or is disjoint, that is there is no demand having the same origin or destination node as another demand.

In all the following tables, each instance is indicated by

- a letter "r" or "d" which indicates whether the demand set is rooted or disjoint,
- $|V|$, the number of nodes of the graph,
- $|D|$, the number of demands.

For example, r-30-10 designates an instance whose graph has 30 nodes, $|D| = 10$ and the demand set is rooted. Also, d-48-24 designates an instance whose graph has 48 nodes, $|D| = 15$ and the demand set is disjoint.

During our preliminary tests, we have noticed that for $L = 2$, our Branch-and-Cut algorithm based on the separated flow formulation performs better than the Branch-and-Cut algorithm based on the other formulations. Also, for $L = 3$, our Branch-and-Cut algorithm based on the natural formulation performs better compared to the other formulations. Thus, in this computational study, we have decided to present the results obtained with our Branch-and-Cut algorithms based on

- the separated flow formulation when $L = 2$,
- the natural formulation when $L = 3$.

We have also noticed during the preliminary tests that, for both $L = 2$ and $L = 3$, the double cut, triple path-cut and aggregated cut inequalities have a weak effect on the resolution, while they increase the separation time spent by the Branch-and-Cut algorithms and increase the overall CPU time. Thus, we have decided to remove these three types of inequalities from the Branch-and-Cut algorithms, and we only use the partition inequalities (3.1), (3.2), (3.3), (3.4) and the Steiner-SP-partition inequalities (4.6).

Table 1 summarizes the configurations of the Branch-and-Cut algorithms for each case $L = 2$ and $L = 3$.

6.1 Efficiency of the Branch-and-Cut algorithm for $L = 2$

Our first series of experiments concern the kHNDP for $L = 2$ and $k = 3, 4, 5$. Remember that for the case $L = 2$, our Branch-and-Cut algorithm is based on the separated flow formulation, and we only use the partition inequalities. The instances we have considered have graphs with 21 up to 70 nodes and a number of demands up to 50. The results are summarized in Tables 2, for $k = 3$, $k = 4$ and $k = 5$, respectively. The entries of these tables are described as follows:

$|V|$: number of nodes of the graph;
$|D|$: number of demands.

For each value of k:

NHCP: number of generated Hop-Constrained partition in equalities;
COPT: best upper bound obtained by the Branch-and-Cut algorithm;
GAP: relative error between the final lower bound and the best upper bound;
NSUB: number of nodes generated in the Branch-and-Bound tree;
CPU: total CPU time achieved by our Branch-and-Cut algorithm.

We start by analyzing the results for $k = 3$. We can see that all the instances are solved to optimality by our Branch-and-Cut algorithm. Also, most of the instances are solved in less than 15 min. Only 2 instances over 29 are solved within more than 15 min. We can also see in this table that the partition inequalities are generated for almost all the instances, and this, for both rooted and disjoint demands.

For $k = 4$, we also see that all the instances are solved to optimality, within less than 15 min for most of them. Only 2 instances are solved with more than 15 min. We also notice that the partition inequalities are generated only for rooted demands, while no partition inequalities are used to solve the instances with disjoint demands.

Finally, for $k = 5$, we still solve all the instances to optimality. Here, only 1 instance over 29 are solved with more than 15 min. Also, partition inequalities are generated for almost all the instances, and this, for both rooted and disjoint demands.

We conclude this section by analyzing the efficiency of the partition inequalities in the Branch-and-Cut algorithm. For this, we have run the algorithm without these inequalities and compare the results to the case when they are used. We do this comparison only for $k = 3$ and $L = 2$. The results are summarize in Table 3. For ease of comparison, the CPU time is given in seconds.

The results of Table 3 show that all the instances are still solved to optimality with and without the partition inequalities. Also, it appears that, for most of the instances, the CPU time is slightly better when the partition inequalities are not used. However, for several instances, the CPU time is considerably increased when the partition inequalities are not used. For example, instance r-70-50 is solved in 7604 s when the partition inequalities are used, while it is solved in 14,090 s when they are not used. Similarly, instance r-70-40 is solved in 1177s with partition inequalities and in 1696s without the partition inequalities. It is important to remark that the results shown here are obtained with the cut generation tool of SCPI kept activated, and this, even when our partition inequalities are not used. Clearly, the inequalities generated by SCIP play a role in the efficiency of the algorithm, but the above observations also show that our partition inequalities also have a positive impact on the resolution of the kHNDP when $L = 2$.

6.2 Efficiency of the Branch-and-Cut algorithm for $L = 3$

Our second series of experiments concerns our Branch-and-Cut algorithm for $L = 3$ and $k = 3, 4, 5$. Recall that our algorithm for $L = 3$ is based on the natural formulation. The results are given, for $k = 3, 4, 5$, in Table 4. The entries of the table are

$|V|$: number of nodes of the graph;

$|D|$: number of demands,

For each value of k:

NCUT: number of generated st-cut and L-st-path-cut inequalities;
NP: number of generated Hop-Constrained partition inequalities;
NSP: number of generated Steiner-SP-partition inequalities;
COPT: best upper bound obtained by the Branch-and-Cut algorithm;

Table 1 Configuration of our Branch-and-Cut algorithms for $L = 2$ and $L = 3$

	Formulation	Used valid inequalities	SCIP cut generation tool
$L = 2$	Separated formulation	Partition inequalities	ON
$L = 3$	Natural formulation	Partition inequalities + Steiner-SP-partition inequalities (only for k odd)	OFF

GAP: relative error between the final lower bound and the best upper bound;
NSUB: number of nodes generated in the Branch-and-Bound tree;
CPU: total CPU time achieved by our Branch-and-Cut algorithm.

We first notice that, contrarily to the case where $L = 2$, most of the instances are not solved to optimality within the maximum CPU time allowed (i.e., 5 h). However, for $k = 3$ and for the rooted demands, we notice that the optimality gap is less than 12% for 6 instances over 18 rooted instances. For instances with disjoint demands, the optimality gap is globally much larger, even if, for half of the instances, it is less than 20%. However, three instances, d-48-24, d-70-26 and d-70-35, have a quite large optimality gap (resp. 61.9%, 76.1% and 76.2%). It also appears that for the case of disjoint demands, no partition inequalities are generated while a few number of Steiner-SP-partition inequalities are found during the resolution. This shows that for $k = 3$, the kHNDP is still difficult to solve when the demands are disjoints.

For $k = 4$ and $k = 5$, we make the same observations as for $k = 3$. Indeed, most of the instances are not solved to optimality within the maximum CPU time allowed, while the optimality gap is less than 20% for the majority of the instances, having graphs with less than 70 nodes. We also remark that the partition inequalities are mainly generated only for instances with rooted demands, while none are generated for disjoint demands. Finally, very few Steiner-SP-partition inequalities are generated during the resolution. The results suggest that the partition inequalities are not useful for the case of disjoint demands. However, we believe that our separation algorithm is not enough to detect violated partition inequalities when the demands are disjoint. Thus, an effort should be made in order to devise a more efficient separation algorithm for the partition inequalities with disjoint demands.

As for the case where $L = 2$, we conclude this section by analyzing the efficiency of the valid inequalities we have used in the Branch-and-Cut algorithm for $L = 3$. For this, we have run the algorithm without these inequalities and compare the results to those obtained when they are used. We do this comparison only for $k = 3$ and for the rooted demands. The following table gives the results obtained with and without using the partition and Steiner-SP-partition inequalities.

We start our discussion by the instances which are solved to optimality. It appears from Table 5 that for these instances, except in two cases (over 18), the CPU time is increased when the partition and Steiner-SP-partition inequalities are not used. For the instances which are not solved to optimality within the maximum CPU time, we clearly notice an increase of the optimality gap when the partition and Steiner-SP-partition inequalities are not used. Moreover, the upper bound obtained without the inequalities, except in one case, is higher than when the inequalities are used. All these observations clearly show that for $L = 3$, the partition and Steiner-SP-partition inequalities have a positive effect on the resolution of the kHNDP.

6.3 Comparison against CPLEX Branch-and-Cut and Benders decomposition algorithms

In this section, we compare our Branch-and-Cut algorithm against a resolution using CPLEX Branch-and-Cut framework and a Benders decomposition framework. The two algorithms are based on the separated flow formulation (2.11) which, we recall, is a compact formulation. For the CPLEX Branch-and-Cut algorithm, CPLEX solves each LP and manages itself the Branch-and-Bound tree. For the automatic Benders decomposition framework, CPLEX relies on the type of the variables (integer or continuous) to build the decomposition. Since in the Separated Flow Formulation, the flow variables are continuous, only the design variables are kept by CPLEX in the master problem of the Benders decomposition. For both algorithms, the default settings are used and all the internal stuffs (heuristics, valid inequalities, etc.) are kept activated.

For this comparison, we have run CPLEX Branch-and-Cut algorithm and CPLEX Benders decomposition algorithm for the same instances as for our Branch-and-Cut algorithms. The maximum CPU time is set to 5 h. The comparison is done for both $L = 2$ and $L = 3$, and aims at comparing, for each instance, the CPU time, the upper bound and the final gap achieved by the three algorithms. The entries of the following tables are:

$|V|$: number of nodes of the graph;
$|D|$: number of demands,
CPU_BEN: total CPU time, in seconds, achieved by CPLEX Benders decomposition algorithm;

Table 2 Results of the Branch-and-Cut algorithm for $L = 2$ and $k = 3, 4, 5$ (Separated Flow Formulation)

	$\|V\|$	$\|D\|$	$k = 3$				$k = 4$				$k = 5$			
			NHCP	COPT	GAP	CPU	NHCP	COPT	GAP	CPU	NHCP	COPT	GAP	CPU
r	21	15	2	7138	0	00:00:00	3	8931	0	00:00:01	0	10,805	0	00:00:00
r	21	17	1	7790	0	00:00:00	1	9818	0	00:00:00	2	12,030	0	00:00:00
r	21	20	9	8762	0	00:00:01	1	10,953	0	00:00:00	2	13,288	0	00:00:01
r	30	10	0	8299	0	00:00:00	0	11,538	0	00:00:00	0	14,888	0	00:00:00
r	30	15	1	12,512	0	00:00:00	0	16,235	0	00:00:01	0	20,926	0	00:00:00
r	30	20	2	14,215	0	00:00:01	1	18,011	0	00:00:01	0	22,880	0	00:00:00
r	30	25	6	15,610	0	00:00:03	2	19,830	0	00:00:03	1	24,609	0	00:00:00
r	48	20	2	21,586	0	00:00:01	7	27,476	0	00:00:05	0	34,287	0	00:00:00
r	48	30	11	29,326	0	00:00:38	8	37,118	0	00:00:23	1	44,411	0	00:00:08
r	48	40	29	37,458	0	00:10:21	41	47,305	0	00:03:21	3	56,200	0	00:00:38
r	52	20	0	14,093	0	00:00:01	0	17,887	0	00:00:00	0	22,869	0	00:00:03
r	52	30	14	17,643	0	00:00:12	3	22,545	0	00:00:03	0	27,611	0	00:00:02
r	52	40	39	21,041	0	00:03:53	5	26,633	0	00:01:23	4	31,942	0	00:00:11
r	52	50	98	24,619	0	00:09:32	37	31,457	0	00:10:37	3	37,649	0	00:01:22
r	70	20	14	1314	0	00:00:28	4	1666	0	00:00:10	6	1982	0	00:00:04
r	70	30	46	1798	0	00:04:16	25	2289	0	00:02:22	12	2737	0	00:00:30
r	70	40	25	2180	0	00:19:37	15	2781	0	00:20:27	4	3332	0	00:03:56
r	70	50	34	2590	0	02:06:44	16	3321	0	02:32:32	5	3985	0	00:54:26
r	100	20	2	33,952	0	00:00:04	5	42,650	0	00:00:07	3	52,301	0	00:00:00
r	100	30	25	47,933	0	00:02:34	21	60,665	0	00:01:08	4	72,041	0	00:00:07
r	100	40	74	60,885	0	00:11:28	11	77,542	0	00:05:10	5	92,755	0	00:01:32
r	100	50	60	69,782	0	01:16:16	53	89,528	0	00:40:31	11	107,543	0	00:09:25
r	150	20	6	33,952	0	00:00:10	0	42,599	0	00:00:05	1	52,242	0	00:00:00
r	150	30	29	47,933	0	00:01:40	17	60,665	0	00:01:21	4	72,041	0	00:00:07
r	150	40	59	60,885	0	00:39:55	21	77,542	0	00:06:47	5	92,755	0	00:02:08
r	150	50	56	69,782	0	00:39:54	40	89,528	0	00:44:37	9	107,445	0	00:15:12
d	21	10	0	8313	0	00:00:00	0	10,915	0	00:00:00	0	14,345	0	00:00:01
d	30	10	11	12,124	0	00:00:01	0	16,097	0	00:00:00	0	21,324	0	00:00:01
d	30	15	4	15,868	0	00:00:01	0	21,333	0	00:00:00	0	27,513	0	00:00:03
d	48	15	24	32,097	0	00:00:06	0	42,503	0	00:00:01	1	54,554	0	00:00:05
d	48	20	131	44, 400	0	00:00:22	0	57,508	0	00:00:00	3	74,317	0	00:00:31
d	48	24	122	52,619	0	00:00:27	0	68,370	0	00:00:00	21	88,147	0	00:01:01
d	52	20	15	18,480	0	00:00:09	0	24,586	0	00:00:01	0	31,854	0	00:00:19
d	52	26	8	24,125	0	00:00:12	0	32,175	0	00:00:00	0	42,145	0	00:00:53
d	70	15	16	1605	0	00:00:02	0	2102	0	00:00:00	2	2736	0	00:00:06
d	70	26	103	2420	0	00:00:39	0	3145	0	00:00:00	8	4072	0	00:00:20
d	70	35	172	3342	0	00:01:22	0	4370	0	00:00:00	20	5646	0	00:01:53
d	100	20	56	75,033	0	00:00:11	0	98,022	0	00:00:03	6	126,591	0	00:00:36
d	100	35	332	123,247	0	00:04:01	0	161,156	0	00:00:04	6	207,350	0	00:07:24
d	100	50	634	175,764	0	00:09:59	0	229,682	0	00:00:00	50	294,668	0	00:08:00
d	150	30	92	110,135	0	00:01:13	0	144,206	0	00:00:00	35	185,368	0	00:02:37
d	150	50	922	175,696	0	00:22:13	0	229,274	0	00:00:03	31	294,369	0	00:17:24

Table 3 Results of the Branch-and-Cut algorithm for $L = 2$ and $k = 3$ (separated flow formulation) without partition inequalities

	$\|V\|$	$\|D\|$	COPT	COPT_NO_CTR	GAP	GAP_NO_CTR	CPU	CPU_NO_CTR
r	21	15	7138	7138	0	0	0	0
r	21	17	7790	7790	0	0	0	0
r	21	20	8762	8762	0	0	1	2
r	30	10	8299	8299	0	0	0	0
r	30	15	12,512	12,512	0	0	0	1
r	30	20	14,215	14,215	0	0	1	1
r	30	25	15,610	15,610	0	0	3	4
r	48	20	21,586	21,586	0	0	1	1
r	48	30	29,326	29,326	0	0	38	71
r	48	40	37,458	37,458	0	0	621	799
r	52	20	14,093	14,093	0	0	1	1
r	52	30	17,643	17,643	0	0	12	9
r	52	40	21,041	21,041	0	0	233	213
r	52	50	24,619	24,619	0	0	572	468
r	70	20	1314	1314	0	0	28	16
r	70	30	1798	1798	0	0	256	160
r	70	40	2180	2180	0	0	1177	1696
r	70	50	2590	2590	0	0	7604	14,090
r	100	20	33,952	33,952	0	0	4	5
r	100	30	47,933	47,933	0	0	154	75
r	100	40	60,885	60,885	0	0	688	496
r	100	50	69,782	69,782	0	0	4576	2327
r	150	20	33,952	33,952	0	0	10	8
r	150	30	47,933	47,933	0	0	100	162
r	150	40	60,885	60,885	0	0	2395	915
r	150	50	69,782	69,782	0	0	2394	5342
d	21	10	8313	8313	0	0	0	0
d	30	10	12,124	12,124	0	0	1	1
d	30	15	15,868	15,868	0	0	1	2
d	48	15	32,097	32,097	0	0	6	3
d	48	20	44,400	44,400	0	0	22	19
d	48	24	52,619	52,619	0	0	27	28
d	52	20	18,480	18,480	0	0	9	9
d	52	26	24,125	24,125	0	0	12	8
d	70	15	1605	1605	0	0	2	2
d	70	26	2420	2420	0	0	39	31
d	70	35	3342	3342	0	0	82	43
d	100	20	75,033	75,033	0	0	11	5
d	100	35	123,247	123,247	0	0	241	310
d	100	50	175,764	175,764	0	0	599	736
d	150	30	110,135	110,135	0	0	73	76
d	150	50	175,696	175,696	0	0	1333	1841

Table 4 Results for the Branch-and-Cut algorithm with $L = 3$ and $k = 3, 4, 5$ (natural formulation)

	$\|V\|$	$\|D\|$	$k = 3$					$k = 4$				$k = 5$				
			NHCP	NSP	COPT	GAP	CPU	NHCP	COPT	GAP	CPU	NHCP	NSP	COPT	GAP	CPU
r	21	15	42	1	5472	0	00:00:04	23	7273	0	00:00:06	0	0	9505	0	00:00:08
r	21	17	30	1	5864	0	00:00:11	33	7824	0	00:00:12	9	0	10,048	0	00:00:06
r	21	20	2	0	6466	0	00:00:48	2	8556	0	00:02:03	9	0	10,982	0	00:01:23
r	30	10	58	22	7611	0	00:00:03	24	10,655	0	00:00:05	0	9	13,728	0	00:00:03
r	30	15	85	14	10,109	0	00:00:25	3	13,963	0	00:00:53	0	1	18,278	0	00:01:08
r	30	20	400	33	11,182	0	00:02:31	28	15,041	0	00:01:26	0	2	19,939	0	00:06:22
r	30	25	14,664	329	12,427	0	02:34:26	412	16,268	0	00:19:29	26	0	21,112	0	00:13:31
r	48	20	2065	890	16,684	0	00:56:47	173	22,040	0	01:31:53	1	13	27,818	0	00:28:21
r	48	30	16,511	83	20,988	4.83	05:00:00	5658	27,460	5.23	05:00:00	4	12	34,401	3.69	05:00:05
r	48	40	4065	16	27,109	10.3	05:00:00	673	34,304	8.79	05:00:00	1725	0	41,371	4.14	05:00:00
r	52	20	2599	334	11,154	0	00:58:24	49	14,979	0	02:43:34	0	253	19,236	0	01:57:10
r	52	30	13,358	1051	13,791	6.14	05:00:00	3698	18,035	4.00	05:00:00	0	3	22,831	3.48	05:00:00
r	52	40	5136	1531	16,295	8.14	05:00:00	2244	20,737	4.57	05:00:00	3	1	25,625	2.28	05:00:00
r	52	50	795	3	18,856	9.95	05:00:00	288	24,483	8.01	05:00:00	181	0	30,231	5.23	05:00:00
r	70	20	4811	1318	922	7.04	05:00:00	3988	1204	7.44	05:00:00	407	276	1481	5.5	05:00:00
r	70	30	3338	793	1263	11.6	05:00:00	2466	1666	12.1	05:00:01	15	29	2029	9.25	05:00:00
r	70	40	2142	615	1534	16.2	05:00:00	1441	1955	13.2	05:00:00	178	6	2362	9.75	05:00:00
r	70	50	782	157	1808	18.1	05:00:00	624	2349	17.5	05:00:00	129	1	2825	13.6	05:00:00
d	21	10	0	0	6675	0	00:02:45	0	8929	0	00:11:57	0	0	11,313	0	00:15:15
d	30	10	0	52	10,254	0	00:01:00	0	14,058	0	00:00:37	0	0	18,491	0	00:01:23
d	30	15	0	0	13,246	2.28	05:00:00	0	17,983	2.45	05:00:07	0	0	22,838	1.31	05:00:00
d	48	15	4	52	25,615	14.9	05:00:00	0	33,884	14.5	05:00:00	0	2	41,392	12.1	05:00:00
d	48	20	0	9	37,779	27.1	05:00:00	0	50,104	27.9	05:00:00	0	0	57,263	20.2	05:00:00
d	48	24	0	0	82,891	61.9	05:00:00	0	101,348	58.6	05:00:00	0	0	109,620	51.8	05:00:00
d	52	20	0	208	16,078	7.48	05:00:00	0	22,007	8.4	05:00:00	0	0	28,018	7.92	05:00:00
d	52	26	0	0	20,430	10.9	05:00:00	0	28,381	12.6	05:00:00	0	0	35,408	10.1	05:00:00
d	70	15	0	229	1394	17.6	05:00:00	0	1810	16.1	05:00:00	0	41	2231	13.8	05:00:00
d	70	26	0	136	6726	76.1	05:00:00	0	2871	24.5	05:00:00	0	0	3453	20.4	05:00:00
d	70	35	0	0	8899	76.2	05:00:00	0	7473	61.8	05:00:00	0	0	10,589	66	05:00:00

Table 5 Results for the Branch-and-Cut algorithm with and without the partition and Steiner-SP-partition inequalities

	$\|V\|$	$\|D\|$	COPT	COPT_NO_CTR	GAP	GAP_NO_CTR	CPU	CPU_NO_CTR
r	21	15	5472	5472	0	0	4	11
r	21	17	5864	5864	0	0	11	16
r	21	20	6466	6466	0	0	48	72
r	30	10	7611	7611	0	0	3	5
r	30	15	10,109	10,109	0	0	25	35
r	30	20	11,182	11,182	0	0	151	129
r	30	25	12,427	12,427	0	0	9266	10,839
r	48	20	16,684	16,684	0	0	3407	5390
r	48	30	20,988	21,044	4.83	5.96	18,000	18,000
r	48	40	27,109	27,497	10.3	13	18,000	18,000
r	52	20	11,154	11,154	0	0	3504	3304
r	52	30	13,791	13,916	6.14	7.87	18,000	18,000
r	52	40	16,295	16,362	8.14	9.04	18,000	18,000
r	52	50	18,856	19,232	9.95	13.5	18,000	18,000
r	70	20	922	933	7.04	8.36	18,000	18,000
r	70	30	1263	1280	11.6	13.6	18,000	18,000
r	70	40	1534	1531	16.2	16.6	18,000	18,000
r	70	50	1808	1854	18.1	21.5	18,000	18,000

Results for $L = 3$ and $k = 3$ (natural formulation)

CPU_CPX: total CPU time, in seconds, achieved by CPLEX Branch-and-Cut algorithm;
CPU_SCIP: total CPU time, in seconds, achieved by our Branch-and-Cut algorithm;
UB_BEN: best upper bound obtained by CPLEX Benders decomposition algorithm;
UB_CPX: best upper bound obtained by CPLEX Branch-and-Cut algorithm;
UB_SCIP: best upper bound obtained by our Branch-and-Cut algorithm;
G_BEN: final gap achieved by CPLEX Benders decomposition algorithm;
G_CPX: final gap achieved by CPLEX Branch-and-Cut algorithm;
G_SCIP: final gap achieved by our Branch-and-Cut algorithm.

Notice that, for ease of comparison, the total CPU time in the following tables is given in seconds. Finally, the results given for our Branch-and-Cut algorithm in the following tables are the same as those reported in Sects. 6.1 and 6.2.

6.3.1 Comparison for $L = 2$

We start the comparison for $L = 2$ and $k = 3, 4, 5$. The results are given in Tables 6, 7 and 8.

For $L = 2$ and $k = 3$ (Tables 6), we can see that both our Branch-and-Cut algorithm and CPLEX Branch-and-Cut algorithm are able to solve all the instances to optimality. However, CPLEX Benders decomposition algorithm was not able to solve 17 instances to optimality within the maximum CPU time. Obviously, we notice that the upper bound provided by CPLEX Benders decomposition algorithm is either equal or higher to that obtained by our Branch-and-Cut algorithm and CPLEX Branch-and-Cut algorithm. We also notice that, even for the instances that are solved to optimality by Benders decomposition algorithm, the CPU time achieved is, for several instances, much larger than the CPU time achieved by the two other algorithms. For example, instance r-52-40 is solved 9751 s by Benders decomposition while it solved in 28 and 233 s with the Branch-and-Cut algorithms. Similarly, for instance d-48-20, the Benders decomposition algorithm is solved in 9751 s while it is solved in less than 30 s with the Branch-and-Cut algorithms.

The observations are the same for $L = 2$ with $k = 4$ and $k = 5$ (Tables 7 and 8). The Benders decomposition algorithm is not able to solve all the instances to optimality while the Branch-and-Cut algorithms do. Moreover, for several instances solved to optimality by the Benders decomposition algorithm, the CPU time achieved is much larger than for the Branch-and-Cut algorithms.

Now, we focus on the comparison between our Branch-and-Cut algorithm and CPLEX Branch-and-Cut algorithm. As both algorithms solve all the instances to optimality, the comparison is done on the CPU time. For $L = 2$ and $k = 3$ (Table 6), we observe that CPLEX Branch-and-Cut algorithm solves the instances faster than our algorithm in most cases. Only 5 instances are solved by CPLEX with a CPU time

Table 6 Comparison between the Branch-and-Cut algorithm (Separated Flow Formulation), CPLEX Branch-and-Cut and CPLEX Benders decomposition algorithms for $L = 2$ and $k = 3$

	$\|V\|$	$\|D\|$	CPU_BEN	CPU_CPX	CPU_SCIP	UB_BEN	UB_CPX	UB_SCIP	G_BEN	G_CPX	G_SCIP
r	21	15	1	1	1	7138	7138	7138	0	0	0
r	21	17	1	1	1	7790	7790	7790	0	0	0
r	21	20	1	1	1	8762	8762	8762	0	0	0
r	30	10	1	1	1	8299	8299	8299	0	0	0
r	30	15	2	1	1	12,512	12,512	12,512	0	0	0
r	30	20	2	1	1	14,215	14,215	14,215	0	0	0
r	30	25	3	1	3	15,610	15,610	15,610	0	0	0
r	48	20	18	1	1	21,586	21,586	21,586	0	0	0
r	48	30	311	11	38	29,326	29,326	29,326	0	0	0
r	48	40	18,000	305	621	37,640	37,458	37,458	3.748	0	0
r	52	20	44	1	1	14,093	14,093	14,093	0	0	0
r	52	30	822	3	12	17,643	17,643	17,643	0	0	0
r	52	40	9751	28	233	21,041	21,041	21,041	0	0	0
r	52	50	18,000	290	572	24,720	24,619	24,619	2.896	0	0
r	70	20	7216	3	28	1314	1314	1314	0	0	0
r	70	30	18,000	34	256	1803	1798	1798	4.49	0	0
r	70	40	18,000	1292	1177	2194	2180	2180	6.64	0	0
r	70	50	18,000	9080	7604	2625	2590	2590	10.28	0	0
r	100	20	659	1	4	33,952	33,952	33,952	0	0	0
r	100	30	18,000	26	154	47,973	47,933	47,933	2.51	0	0
r	100	40	18,000	347	688	61,268	60,885	60,885	6.54	0	0
r	100	50	18,000	3363	4576	70,007	69,782	69,782	7.3	0	0
r	150	20	18,000	1	10	34,085	33,952	33,952	2.75	0	0
r	150	30	18,000	46	100	49,428	47,933	47,933	12.31	0	0
r	150	40	18,000	694	2395	61,127	60,885	60,885	4.42	0	0
r	150	50	18,000	6970	2394	71,726	69,782	69,782	25.85	0	0
d	21	10	1	1	1	8313	8313	8313	0	0	0
d	30	10	1	1	1	12,124	12,124	12,124	0	0	0
d	30	15	2	1	1	15,868	15,868	15,868	0	0	0
d	48	15	41	1	6	32,097	32,097	32,097	0	0	0
d	48	20	3301	3	22	44,400	44,400	44,400	0	0	0
d	48	24	9751	3	27	52,619	52,619	52,619	0	0	0
d	52	20	109	1	9	18,480	18,480	18,480	0	0	0
d	52	26	562	1	12	24,125	24,125	24,125	0	0	0
d	70	15	70	1	2	1605	1605	1605	0	0	0
d	70	26	1425	4	39	2420	2420	2420	0	0	0
d	70	35	18,000	11	82	3355	3342	3342	1.27	0	0
d	100	20	4314	2	11	75,033	75,033	75,033	0	0	0
d	100	35	18,000	120	241	130,333	123,247	123,247	7.18	0	0
d	100	50	18,000	671	599	200,674	175,764	175,764	14.15	0	0
d	150	30	18,000	10	73	112,399	110,135	110,135	3.37	0	0
d	150	50	18,000	1910	1333	197,892	175,696	175,696	13.07	0	0

Table 7 Comparison between the Branch-and-Cut algorithm (Separated Flow Formulation), CPLEX Branch-and-Cut and CPLEX Benders decomposition algorithms for $L = 2$ and $k = 4$

	$\|V\|$	$\|D\|$	CPU_BEN	CPU_CPX	CPU_SCIP	UB_BEN	UB_CPX	UB_SCIP	G_BEN	G_CPX	G_SCIP
r	21	15	1	1	1	8931	8931	8931	0	0	0
r	21	17	1	1	1	9818	9818	9818	0	0	0
r	21	20	1	1	1	10,953	10,953	10,953	0	0	0
r	30	10	1	1	1	11,538	11,538	11,538	0	0	0
r	30	15	1	1	1	16,235	16,235	16,235	0	0	0
r	30	20	1	1	1	18,011	18,011	18,011	0	0	0
r	30	25	1	1	3	19,830	19,830	19,830	0	0	0
r	48	20	8	1	5	27,476	27,476	27,476	0	0	0
r	48	30	54	3	23	37,118	37,118	37,118	0	0	0
r	48	40	8937	66	201	47,305	47,305	47,305	0	0	0
r	52	20	7	1	1	17,887	17,887	17,887	0	0	0
r	52	30	64	1	3	22,545	22,545	22,545	0	0	0
r	52	40	282	13	83	26,633	26,633	26,633	0	0	0
r	52	50	11,890	165	637	31,457	31,457	31,457	0	0	0
r	70	20	2318	2	10	1666	1666	1666	0	0	0
r	70	30	18,000	19	142	2289	2289	2289	1.232	0	0
r	70	40	18,000	515	1227	2796	2781	2781	5.587	0	0
r	70	50	18,000	9369	9152	3343	3321	3321	7.472	0	0
r	100	20	70	1	7	42,650	42,650	42,650	0	0	0
r	100	30	10,993	12	68	60,665	60,665	60,665	0	0	0
r	100	40	18,000	177	310	77,708	77,542	77,542	4.005	0	0
r	100	50	18,000	1671	2431	90,074	89,528	89,528	6.347	0	0
r	150	20	114	1	5	42,599	42,599	42,599	0	0	0
r	150	30	15,554	18	81	60,665	60,665	60,665	0	0	0
r	150	40	18,000	280	407	79,007	77,542	77,542	10.02	0	0
r	150	50	18,000	4625	2677	90,294	89,528	89,528	8.21	0	0
d	21	10	1	1	1	10,915	10,915	10,915	0	0	0
d	30	10	1	1	1	16,097	16,097	16,097	0	0	0
d	30	15	1	1	1	21,333	21,333	21,333	0	0	0
d	48	15	3	1	1	42,503	42,503	42,503	0	0	0
d	48	20	5	1	1	57,508	57,508	57,508	0	0	0
d	48	24	9	1	1	68,370	68,370	68,370	0	0	0
d	52	20	2	1	1	24,586	24,586	24,586	0	0	0
d	52	26	6	1	1	32,175	32,175	32,175	0	0	0
d	70	15	3	1	1	2102	2102	2102	0	0	0
d	70	26	4	1	1	3145	3145	3145	0	0	0
d	70	35	36	1	1	4370	4370	4370	0	0	0
d	100	20	29	1	3	98,022	98,022	98,022	0	0	0
d	100	35	584	1	4	161,156	161,156	161,156	0	0	0
d	100	50	1058	1	1	229,682	229,682	229,682	0	0	0
d	150	30	423	1	1	144,206	144,206	144,206	0	0	0
d	150	50	5280	1	3	229,274	229,274	229,274	0	0	0

Table 8 Comparison between the Branch-and-Cut algorithm (Separated Flow Formulation), CPLEX Branch-and-Cut and CPLEX Benders decomposition algorithms for $L = 2$ and $k = 5$

	$\|V\|$	$\|D\|$	CPU_BEN	CPU_CPX	CPU_SCIP	UB_BEN	UB_CPX	UB_SCIP	G_BEN	G_CPX	G_SCIP
r	21	15	1	1	1	10,805	10,805	10,805	0	0	0
r	21	17	1	1	1	12,030	12,030	12,030	0	0	0
r	21	20	1	1	1	13,288	13,288	13,288	0	0	0
r	30	10	1	1	1	14,888	14,888	14,888	0	0	0
r	30	15	1	1	1	20,926	20,926	20,926	0	0	0
r	30	20	1	1	1	22,880	22,880	22,880	0	0	0
r	30	25	1	1	1	24,609	24,609	24,609	0	0	0
r	48	20	6	1	1	34,287	34,287	34,287	0	0	0
r	48	30	20	2	8	44,411	44,411	44,411	0	0	0
r	48	40	164	14	38	56,200	56,200	56,200	0	0	0
r	52	20	7	1	3	22,869	22,869	22,869	0	0	0
r	52	30	20	1	2	27,611	27,611	27,611	0	0	0
r	52	40	31	3	11	31,942	31,942	31,942	0	0	0
r	52	50	106	16	82	37,649	37,649	37,649	0	0	0
r	70	20	829	2	4	1982	1982	1982	0	0	0
r	70	30	9745	5	30	2737	2737	2737	0	0	0
r	70	40	18,000	54	236	3338	3332	3332	2.096	0	0
r	70	50	18,000	1450	3266	3998	3985	3985	4.286	0	0
r	100	50	32	1	1	52,301	52,301	52,301	0	0	0
r	100	20	258	2	7	72,041	72,041	72,041	0	0	0
r	100	30	13,728	18	92	92,755	92,755	92,755	0	0	0
r	100	40	18,000	130	565	107,826	107,543	107,543	3.257	0	0
r	150	50	67	1	1	52,242	52,242	52,242	0	0	0
r	150	30	183	2	7	72,041	72,041	72,041	0	0	0
r	150	40	8014	28	128	92,755	92,755	92,755	0	0	0
r	150	50	18,000	276	912	108,598	107,445	107,445	6.392	0	0
d	21	10	1	1	1	14,345	14,345	14,345	0	0	0
d	30	10	1	1	1	21,324	21,324	21,324	0	0	0
d	30	15	2	1	3	27,513	27,513	27,513	0	0	0
d	48	15	10	1	5	54,554	54,554	54,554	0	0	0
d	48	20	1211	4	31	74,317	74,317	74,317	0	0	0
d	48	24	5289	6	61	88,147	88,147	88,147	0	0	0
d	52	20	70	1	19	31,854	31,854	31,854	0	0	0
d	52	26	5071	9	53	42,145	42,145	42,145	0	0	0
d	70	15	87	1	6	2736	2736	2736	0	0	0
d	70	26	1847	4	20	4072	4072	4072	0	0	0
d	70	35	18,000	16	113	5666	5646	5646	1.178	0	0
d	100	20	18,000	6	36	126,591	126,591	126,591	0.341	0	0
d	100	35	18,000	314	444	209,992	207,350	207,350	2.283	0	0
d	100	50	18,000	1297	480	306,833	294,668	294,668	4.965	0	0
d	150	30	18,000	103	157	188,740	185,368	185,368	2.779	0	0
d	150	50	18,000	1907	1044	317,405	294,369	294,369	8.308	0	0

Table 9 Comparison between the Branch-and-Cut algorithm (Natural Formulation), CPLEX Branch-and-Cut and CPLEX Benders decomposition algorithms for $L = 3$ and $k = 3$

	$\|V\|$	$\|D\|$	CPU_BEN	CPU_CPX	CPU_SCIP	UB_BEN	UB_CPX	UB_SCIP	G_BEN	G_CPX	G_SCIP
r	21	15	2	15	4	5472	5472	5472	0	0	0
r	21	17	5	26	11	5864	5864	5864	0	0	0
r	21	20	90	394	48	6466	6466	6466	0	0	0
r	30	10	1	11	3	7611	7611	7611	0	0	0
r	30	15	14	77	25	10,109	10,109	10,109	0	0	0
r	30	20	71	759	151	11,182	11,182	11,182	0	0	0
r	30	25	8462	18,000	9266	12,427	12,427	12,427	0	2.86	0
r	48	20	5130	9868	3407	16,684	16,684	16,684	0	0	0
r	48	30	18,000	18,000	18,000	21,981	21,024	20,988	12.21	8.33	4.83
r	48	40	18,000	18,000	18,000	28,300	27,087	27,109	18.39	15.21	10.3
r	52	20	18,000	7946	3504	11,154	11,154	11,154	0.54	0	0
r	52	30	18,000	18,000	18,000	13,949	13,743	13,791	8.09	8.30	6.14
r	52	40	18,000	18,000	18,000	16,985	16,210	16,295	14.73	10.63	8.14
r	52	50	18,000	18,000	18,003	19,559	18,767	18,856	16.36	14.03	9.95
r	70	20	18,000	18,000	18,000	1002	932	922	18.16	11.06	7.04
r	70	30	18,000	18,000	18,000	1397	1278	1263	25.29	18.23	11.6
r	70	40	18,000	18,000	18,000	2084	1555	1534	41.86	23.56	16.2
r	70	50	18,000	18,000	18,000	2182	1819	1808	37.27	25.13	18.1
d	21	10	261	152	165	6675	6675	6675	0	0	0
d	30	10	82	109	60	10,254	10,254	10,254	0	0	0
d	30	15	18,000	18,000	18,000	13,387	13,246	13,246	4.46	2.13	2.28
d	48	15	18,000	18,000	18,000	29,692	24,366	25,615	28.36	10.64	14.9
d	48	20	18,000	18,000	18,000	44,128	32,957	37,779	38.65	16.9	27.1
d	48	24	18,000	18,000	18,000	52,909	39,216	82,891	40.14	18.82	61.9
d	52	20	18,000	18,000	18,000	17,061	15,620	16,078	13.66	5.81	7.48
d	52	26	18,000	18,000	18,000	22,751	20,677	20,430	20.27	12.56	10.9
d	70	15	18,000	18,000	18,000	1643	1290	1394	31.63	13.23	17.6
d	70	26	18,000	18,000	18,000	2663	2009	6726	39.44	19.99	76.1
d	70	35	18,000	18,000	18,000	4863	3078	8899	56.05	30.36	76.2

larger than with our Branch-and-Cut algorithm. The observation is the same for $L = 2$ and $k = 4$ and $k = 5$. CPLEX Branch-and-Cut algorithm provides, in general, better CPU Time than our Branch-and-Cut algorithm.

6.3.2 Comparison for $L = 3$

We finish this section by comparing now the three resolution algorithms for the case where $L = 3$. As before, we compare the CPU time, the quality of the upper bound and the final gap obtained by the three algorithms. The results are given in Tables 9, 10 and 11.

We start our analysis by $L = 3$ and $k = 3$ (Table 9). First, we notice that all the algorithms solve only a few number of instances to optimality. For most of the instances, the three algorithms spend more than 5 h to obtain the upper bound. However, we can find some instances where each algorithm outperforms the others. For example with instance r-30–25, the three algorithms provide the optimal solution but CPLEX Branch-and-Cut algorithm is not able to prove the optimality within the time limit, while the Benders decomposition algorithm and our Branch-and-Cut algorithm do. Also, for instance r-52–20, the three algorithms have found the optimal solution but only our Branch-and-Cut algorithm and CPLEX Branch-and-Cut algorithm are able to prove the optimality before 5 h. We also observe that for this later instance the CPU time used by our algorithm is more than half smaller than the CPU time achieved by the Branch-and-Cut algorithm of CPLEX.

Except these few instances, all the other instances are not solved to optimality by all the three algorithms. Thus, for our purpose, we compare the three algorithms in terms of quality of upper bound. For this, we introduce the so-called "Quality Gap" which aims at measuring the ability of an algorithm to

Table 10 Comparison between the Branch-and-Cut algorithm (Natural Formulation), CPLEX Branch-and-Cut and CPLEX Benders decomposition algorithms for $L = 3$ and $k = 4$

	$\|V\|$	$\|D\|$	CPU_BEN	CPU_CPX	CPU_SCIP	UB_BEN	UB_CPX	UB_SCIP	G_BEN	G_CPX	G_SCIP
r	21	15	2	10	6	7273	7273	7273	0	0	0
r	21	17	4	19	12	7824	7824	7824	0	0	0
r	21	20	35	221	123	8556	8556	8556	0	0	0
r	30	10	4	7	5	10,655	10,655	10,655	0	0	0
r	30	15	47	72	53	13,963	13,963	13,963	0	0	0
r	30	20	44	250	86	15,041	15,041	15,041	0	0	0
r	30	25	824	3895	1169	16,268	16,268	16,268	0	0	0
r	48	20	7829	18,000	5513	22,040	22,040	22,040	0	1.396	0
r	48	30	18,000	18,000	18,000	28,025	27,555	27,460	7.615	7.508	5.23
r	48	40	18,000	18,000	18,000	35,046	33,887	34,304	11.98	9.389	8.79
r	52	20	8368	18,000	9814	14,979	15,016	14,979	0	2.129	0
r	52	30	18,000	18,000	18,000	18,183	18,116	18,035	4.675	5.862	4
r	52	40	18,000	18,000	18,000	21,296	20,898	20,737	8.147	7.268	4.57
r	52	50	18,000	18,000	18,000	25,064	24,155	24,483	11.45	8.936	8.01
r	70	20	18,000	18,000	18,000	1283	1191	1204	15.51	7.858	7.44
r	70	30	18,000	18,000	18,000	1869	1647	1666	24.12	15.49	12.1
r	70	40	18,000	18,000	18,000	2532	1977	1955	35.56	18.15	13.2
r	70	50	18,000	18,000	18,000	2628	2336	2349	29.18	20.75	17.5
d	21	10	1399	205	717	8929	8929	8929	0	0	0
d	30	10	54	26	37	14,058	14,058	14,058	0	0	0
d	30	15	18,000	18,000	18,000	18,034	17,926	17,983	3.311	2.059	2.45
d	48	15	18,000	18,000	18,000	39,274	32,328	33,884	27.53	11.66	14.5
d	48	20	18,000	18,000	18,000	56,179	44,294	50,104	35.85	18.03	27.9
d	48	24	18,000	18,000	18,000	64,823	52,313	101,348	35.03	19.54	58.6
d	52	20	18,000	18,000	18,000	23,311	21,187	22,007	13.96	6.101	8.4
d	52	26	18,000	18,000	18,000	32,438	27,705	28,381	23.85	11.14	12.6
d	70	15	18,001	18,000	18,000	2290	1737	1810	34.12	13.13	16.1
d	70	26	18,000	18,000	18,000	3159	2715	2871	31.29	20.28	24.5
d	70	35	18,000	18,000	18,000	5765	9450	7473	50.23	69.56	61.8

produce upper bound close to the best know upper bound. If, for a given instance, UB_{ref} is the best known upper bound, and UB_i is the upper bound obtained by an algorithm i for that instance, then the quality gap measured for formulation i is given by

$$QGap_i = 100 * \frac{UB_i - UB_{ref}}{UB_{ref}}.$$

We measure the quality gap for all the instances and compare the algorithms by comparing the average quality gap over all the instances, and this, for $L = 3$ and every $k \in \{3, 4, 5\}$. Notice that an average quality gap close to 0 indicates that the concerned algorithm gives, in most cases, an upper bound close to the best know solution.

In order to have a relevant interpretation of this average quality gap, we distinguish the case of rooted and disjoint demands. Table 12 gives the average quality gaps obtained for rooted instances only, $L = 3$ and $k = 3, 4, 5$.

We can see from Table 12 that for $k = 3$, our Branch-and-Cut algorithm has the better quality gap. This means that for $k = 3$ and for most of the instances with rooted demands, our algorithm produces better solutions than CPLEX Branch-and-Cut and Benders decomposition algorithms. For $k = 4$ and $k = 5$, we notice that CPLEX Branch-and-Cut algorithm has the better quality gap, however our Branch-and-Cut algorithm also show a small quality gap in this case. This means that both CPLEX Branch-and-Cut algorithm and our Branch-and-Cut algorithm have similar overall performance, with an advantage to CPLEX Branch-and-Cut algorithm. Finally, we notice that the quality gap achieved for CPLEX automatic Benders decomposition algorithm is quite large compared to the two other algorithms, and this for all $k \in \{3, 4, 5\}$. This

Table 11 Comparison between the Branch-and-Cut algorithm (Natural Formulation), CPLEX Branch-and-Cut and CPLEX Benders decomposition algorithms for $L = 3$ and $k = 5$

	$\|V\|$	$\|D\|$	CPU_BEN	CPU_CPX	CPU_SCIP	UB_BEN	UB_CPX	UB_SCIP	G_BEN	G_CPX	G_SCIP
r	21	15	3	44	8	9505	9505	9505	0	0	0
r	21	17	2	35	6	10,048	10,048	10,048	0	0	0
r	21	20	11	486	83	10,982	10,982	10,982	0	0	0
r	30	10	2	12	3	13,728	13,728	13,728	0	0	0
r	30	15	64	573	68	18,278	18,278	18,278	0	0	0
r	30	20	261	3269	382	19,939	19,939	19,939	0	0	0
r	30	25	435	7903	811	21,112	21,112	21,112	0	0	0
r	48	20	2127	15,396	1701	27,818	27,818	27,818	0	0	0
r	48	30	18,000	18,000	18,000	34,854	34,509	34,401	5.28	5.88	3.69
r	48	40	17,174	18,000	18,000	42,441	41,942	41,371	9.19	7.17	4.14
r	52	20	16,619	18,000	7030	19,236	19,236	19,236	0	1.37	0
r	52	30	18,000	18,000	18,000	22,801	22,824	22,831	3.07	4/61	3.48
r	52	40	18,000	18,000	18,000	26,097	25,933	25,625	4.30	4.60	2.28
r	52	50	18,000	18,000	18,000	29,879	29,849	30,231	4.43	5.64	5.23
r	70	20	18,000	18,000	18,000	1532	1453	1481	10.11	4.95	5.5
r	70	30	18,000	18,000	18,000	2145	2007	2029	15.74	10.72	9.25
r	70	40	18,000	18,000	18,000	2548	2358	2362	17.75	12.12	9.75
r	70	50	18,000	18,000	18,000	3182	2777	2825	24.93	14.37	13.6
d	21	10	1883	1349	915	11,313	11,313	11,313	0	0	0
d	30	10	47	289	83	18,491	18,491	18,491	0	0	0
d	30	15	18,000	18,000	18,000	22,838	22,838	22,838	1516	1643	1.31
d	48	15	18,000	18,000	18,000	45,295	40,282	41,392	2059	1075	12.1
d	48	20	18,000	18,000	18,000	60,733	53,790	57,263	2539	1558	20.2
d	48	24	18,000	18,000	18,000	80,531	64,123	109,620	3437	1767	51.8
d	52	20	18,000	18,000	18,000	28,803	26,958	28,018	1084	5003	7.92
d	52	26	18,000	18,000	18,000	38,948	35,454	35,408	1843	1056	10.1
d	70	15	18,000	18,000	18,000	2737	2158	2231	3018	1145	13.8
d	70	26	18,000	18,000	18,000	4380	3416	3453	3716	1958	20.4
d	70	35	18,000	18,000	18,000	6720	11,545	10,589	4606	6856	66

Table 12 Average quality gaps for $L = 3$ and rooted demands and $k = 3, 4, 5$

	Av_QGap_BEN	Av_QGap_CPX	Av_QGap_SCIP
$k = 3$	5.31	0.25	0.08
$k = 4$	4.22	0.16	0.30
$k = 5$	2.26	0.17	0.35

Table 13 Average quality gaps for $L = 3$ and disjoint demands

	Av_QGap_BEN	Av_QGap_CPX	Av_QGap_SCIP
$k = 3$	29.35	3.35	97.78
$k = 4$	13.47	5.81	14.35
$k = 5$	11.16	6.54	13.28

clearly shows that this latter algorithm is outperformed by the Branch-and-Cut algorithms (ours and that of CPLEX).

Now, we turn our attention to the case of disjoint demands. Table 13 gives the average quality gap for instances with disjoint demands, $L = 3$ and $k = 3, 4, 5$.

Contrarily to the case of rooted demands, the average quality gaps for all the algorithms are more distant from 0. This means that each algorithm may have some set of instances for which it provides poor solutions w.r.t. the others. However, CPLEX Branch-and-Cut algorithm still has a smaller average quality gap than the other algorithms. We also notice that for $k = 4$ and $k = 5$, CPLEX Benders decomposition algorithms and our Branch-and-Cut algorithm have very similar average gaps, which means that both algorithms may provide similar quality of solutions. However, it is important to remark that our Branch-and-Cut algorithm has not generated partition inequalities and very few Steiner-SP-partition inequalities

have been generated for instances with disjoint demands. As mentioned before, we believe that this is due to the difficulty of devising an efficient separation algorithm for partition inequalities with $L = 3$ and disjoint demands. Thus, our Branch-and-Cut algorithm in this latter case reduces to simply solving each linear program (with the separation of st-cut and L-st-path-cut inequalities) and then branching if necessary. No additional (or very few) valid inequalities are added to reinforce the linear relaxation of the natural formulation. Despite this, we see that our algorithm provide overall similar quality of solution than CPLEX Benders decomposition algorithm. Thus, we believe that, with an efficient separation algorithm for partition inequalities, our Branch-and-Cut algorithm will outperform CPLEX Benders decomposition algorithm for instances with disjoint demands.

7 Concluding remarks

In this paper, we have studied the kHNDP and particularly focused on the polytope associated with the problem. We have introduced several classes of inequalities that are valid for the polytope of the problem. Then, we have devised Branch-and-Cut algorithms based on the separated flow formulation and on the natural formulation, and using the valid inequalities, we have introduced in this work.

The computational study we have conducted has shown that the Hop-constrained partition inequalities we have introduced are effective in solving the kHNDP, especially for $L = 3$ and instances with rooted demands. Also, our Branch-and-Cut algorithm is able to solve to optimality instances with large size graphs and large number of demands when $L = 2$. Moreover, the results show that the kHNDP is much harder for $L = 3$ and that our Branch-and-Cut algorithm is able to produce feasible solutions with relatively small optimality gap for this case.

Our study also shows that on one hand, both our Branch-and-Cut algorithm and the Branch-and-Cut framework of CPLEX have good overall performances for both $L \in \{2, 3\}$. On the other hand, it appears that the automatic Benders decomposition framework of CPLEX provides poor results compared to the Branch-and-Cut algorithms (ours and that of CPLEX). This latter observation suggests that a Branch-and-Cut algorithm may be more efficient for solving the kHNDP.

The results presented in this work also point out the fact that the kHNDP when $L = 3$ is a challenging task which still deserves more attention. Moreover, the case of disjoint demands still needs to be efficiently handled. This could be done by first investigating an efficient separation procedure for the partition inequalities for this latter case.

Finally, it appears that the polytope associated with the natural formulation of the kHNDP remains not well known when $L \geq 4$. To the best of our knowledge, except Huygens and Mahjoub (2007) who explicitly described the inequalities which are necessary in the natural formulation of the kHNDP for $L = 4$ and $k = 2$, no further class of valid inequalities is known for that formulation when $L \geq 4$ and $k \geq 2$. Thus, identifying new classes of facet-defining inequalities for any $L \geq 4$ could represent an interesting challenge from a theoretical point of view but also for practical purposes, as these inequalities may help in efficiently solving the kHNDP for any $L \geq 4$.

Author Contributions All the authors contributed to the study conception and design.

Funding The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Data availability The datasets generated during and/or analyzed during the current study are available in the OSF repository, https://osf.io/rbqxn.

Declarations

Conflict of interest All the authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Arslan O, Jabali O, Laporte G (2020) A flexible, natural formulation for the network design problem with vulnerability constraints. INFORMS J Comput 32(1):1–198

Balakrishnana A, Vad Karsten C (2017) Container shipping service selection and cargo routing with transshipment limits. Eur J Oper Res 263:652–663

Bendali F, Diarrassouba I, Mahjoub AR, Mailfert J (2010) On the k-edge-disjoint 3-hop-constrained paths polytope. Discrete Optim 7(1):1–191

Botton Q, Fortz B, Gouveia L, Poss M (2013) Benders decomposition for the hop-constrained survivable network design problem. INFORMS J Comput 25(1):1–191

Chopra S (1994) The k-edge connected spanning subgraph polyhedron. SIAM J Discrete Math 7:245–259

Coullard CR, Gamble AB, Lui J (1994) The k-walk polyhedron. In: Du D-Z, Sun J (eds) Advances in optimization and approximation, nonconvex optimization application 1. Kluwer Academic Publishers, Dordrecht, pp 9–29

Cplex 12.9, ibm (2019) http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Dahl G (1999) The 2-hop spanning tree problem. Oper Res Lett 23((1–2)):21–26

Dahl G (1999) Notes on polyhedra associated with hop-constrained paths. Oper Res Lett 25(2):97–100

Dahl G, Gouveia L (2004) On the directed hop-constrained shortest path problem. Oper Res Lett 32(1):15–22

Dahl G, Johannessen B (2004) The 2-path network problem. Networks 43(3):190–199

Dahl G, Huygens D, Mahjoub AR, Pesneau P (2006) On the k edge-disjoint 2-hop-constrained paths polytope. Oper Res Lett 34(5):577–582

Diarrassouba I (2009) Survivable network design problems with high survivability requirements. Ph.D. thesis, Université Blaise Pascal - Clermond-Ferrand II

Diarrassouba I, Gabrel V, Gouveia L, Mahjoub AR, Pesneau P (2016) Integer programming formulations for the k-edge-connected 3-hop-constrained network design problem. Networks 67(2):148–169

Diarrassouba I, Mahjoub AR (2023) Polyhedral investigation of the *k*-edge-disjoint hop-constrained network design problem. Technical Report HAL, Id: hal-04051494

Didi Biha M, Mahjoub AR (1996) *k*-edge connected polyhedra on series-parallel graphs. Oper Res Lett 19:71–78

Fortz B, Labbe M, Maffioli F (2000) Solving the two-connected network with bounded meshes problem. Oper Res Lett 48:866–877

Fortz B, Mahjoub AR, McCormick ST, Pesneau P (2006) Two-edge connected subgraphs with bounded rings: polyhedral results and branch-and-cut. Math Programm 105(1):85–111

Gerald G, Daniel A, Ksenia B, Wei-Kun C, Leon E, Maxime G, Patrick G, Ambros G, Leona G, Katrin H, Gregor H, Christopher H, Thorsten K, Bodic Pierre L, Maher Stephen J, Frederic M, Matthias M, Erik M, Benjamin M, Pfetsch Marc E, Franziska S, Felipe S, Yuji S, Christine T, Stefan V, Fabian W, Dieter W, Jakob W (2020) The scip optimization suite 7.0. technical report—optimization online

Gouveia L (1999) Multicommodity flow models for spanning trees with hop constraints. Oper Res Lett 25(2):97–100

Gouveia L, Requejo C (2001) A new lagrangean relaxation approach for the hop-contrained minimum spanning tree problem. Eur J Oper Res 25(2):539–552

Huygens D, Mahjoub AR (2007) Integer programming formulation for the two 4-hop-constrained paths problem. Networks 49(2):135–1442

Huygens D, Mahjoub AR, Pesneau P (2004) Two edge-disjoint hop-constrained paths and polyhedra. SIAM J Discrete Math 18(2):287–312

Huygens D, Labbe M, Mahjoub AR, Pesneau P (2007) The two edge-connected hop-constrained network design problem: valid inequalities and branch-and-cut. Networks 49(1):116–133

Ko C-W, Monma CL (1989) Heuristics for designing highly survivable communication networks. Technical report, Bellcore, Morristown, New Jersey

Lhomme S (2016) Vulnerability and resilience of ports and maritime networks to cascading failures and targeted attacks. In: Ducruet C (ed) Maritime networks. Routledge, London, pp 229–241

Mahjoub AR, Simonetti L, Uchoa E (2013) Hop-level flow formulation for the survivable network design with hop constraints problem. Networks 61(2):171–179

Mahjoub AR, Poss M, Simonetti L, Uchoa E (2019) Distance transformation for network design problems. SIAM J Optim 29(2):1687–1713

Tsplib (1995). http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/