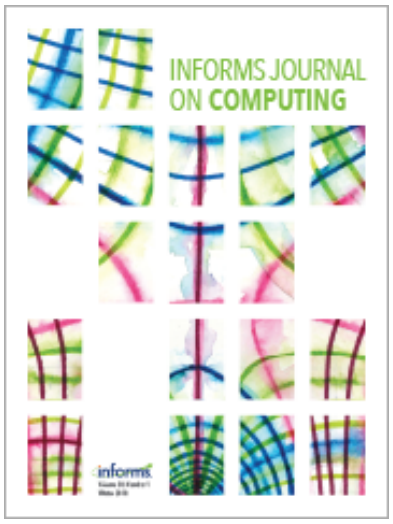

INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Lateness Minimization in Pairwise Connectivity Restoration Problems

Igor Averbakh, Jordi Pereira

To cite this article:
Igor Averbakh, Jordi Pereira (2018) Lateness Minimization in Pairwise Connectivity Restoration Problems. INFORMS Journal on Computing 30(3):522-538. https://doi.org/10.1287/ijoc.2017.0796

http://pubsonline.informs.org/journal/ijoc/

INFORMS JOURNAL ON COMPUTING
Vol. 30, No. 3, Summer 2018, pp. 522–538
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Lateness Minimization in Pairwise Connectivity Restoration Problems

Igor Averbakh,[a] Jordi Pereira[b]

[a] Department of Management, University of Toronto Scarborough, Toronto, Ontario M1C 1A4, Canada; [b] Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Viña del Mar 2581793, Chile

Contact: averbakh@utsc.utoronto.ca, http://orcid.org/0000-0001-6168-4424 (IA); jorge.pereira@uai.cl (JP)

Received: May 10, 2016
Revised: April 22, 2017; August 14, 2017; September 23, 2017
Accepted: October 3, 2017
Published Online: September 21, 2018

https://doi.org/10.1287/ijoc.2017.0796

Abstract. A network is given whose edges need to be constructed (or restored after a disaster). The lengths of edges represent the required construction/restoration times given available resources, and one unit of length of the network can be constructed per unit of time. All points of the network are accessible for construction at any time. For each pair of vertices, a due date is given. It is required to find a construction schedule that minimizes the maximum lateness of all pairs of vertices, where the lateness of a pair is the difference between the time when the pair becomes connected by an already constructed path and the pair's due date. We introduce the problem and analyze its structural properties, present a mixed-integer linear programming formulation, develop a number of lower bounds that are integrated in a branch-and-bound algorithm, and discuss results of computational experiments both for instances based on randomly generated networks and for instances based on 2010 Chilean earthquake data.

History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms.
Funding: The research of Igor Averbakh was supported by the Discovery Grant [238234-2012-RGPIN] from the Natural Sciences and Engineering Research Council of Canada (NSERC).
Supplemental Material: The online appendix is available at https://doi.org/10.1287/ijoc.2017.0796.

Keywords: combinatorial optimization • networks: scheduling • programming: branch and bound • network restoration • network construction • integrated network design and scheduling

1. Introduction

Recently, an increasing amount of research effort has been devoted to *network construction problems*, also known as *integrated network design and scheduling problems*. In these problems, given information about a transportation network that needs to be constructed (or restored after some extreme event/disaster), it is required to find an optimal schedule of construction activities that optimizes the performance of the system over the construction period. Construction/restoration of transportation networks can be quite lengthy and some parts of the network often need to become operational sooner than the whole network is constructed, especially in emergency situations; hence, this class of problems is important.

In this paper, we introduce and study the following problem. There is a network whose edges need to be constructed. Because of limited resources, the construction speed is fixed: one unit of length of the network can be constructed per unit of time. At any time, construction can be conducted at any point of the network; that is, construction resources (workers, machinery, materials) can reach any point of the network in negligible time by some means of transportation that do not use the network. For each pair of vertices, a due date is given. It is required to find a construction schedule that minimizes the maximum lateness of all pairs of vertices, where the lateness of a pair is the difference between the time when the pair becomes connected by an already constructed path and the pair's due date.

One possible application context is restoration of a transportation network where some edges have been damaged as a result of a disaster, which destroyed connectivity. In planning restoration activities with limited resources, it would be natural to prioritize restoring the connectivity, and to take into account that restoring connection may have different urgency for different pairs of vertices, which can be reflected by a choice of the corresponding due dates. For example, a pair of vertices that contain plants that need each other's production for functioning, or a pair of vertices where one vertex contains a fire station or a hospital that serves the population at the other vertex, would need to be connected more urgently than two population centers with little interdependence. Then, the problem of scheduling restoration activities to minimize the maximum lateness can be formulated as our problem on an artificial restoration network whose vertices are the connected components of the survived transportation network and whose edges are the damaged edges of the transportation network, with lengths that represent the corresponding estimated restoration efforts/time. More details will be provided in Section 9.4 where

we discuss experimental results for instances based on 2010 Chilean earthquake data.

Let us discuss justification for the assumption that all points of the network are immediately accessible for construction at any time. First, there are many practical settings where this assumption holds, for example, if the network under construction is embedded into another transportation network that can be used for transporting construction resources but cannot be used for the purposes that the network under construction should serve. For example, streetcar networks, rapid transit networks, water and gas pipeline networks, and underground transportation networks in cities are often embedded into the street network; construction of commercial railways is often done along existing local automobile roads; etc. Second, we use this assumption as an idealized simple way to define the model and the corresponding combinatorial optimization problem, which in fact are applicable also in less restrictive situations. In particular, all results are applicable to the setting where instead of immediate accessibility of any point of the network at any time, we request only that each vertex is accessible at any time. More specifically, as we show in Section 3, we get an equivalent problem if instead of immediate accessibility of any point of the network, we require only that any vertex of the network can always be reached from any other vertex in negligible time by some modes of transportation that do not use the network under construction. Since network construction times are typically large, we believe it is reasonable to assume that transportation times by any common mode of transportation are negligible with respect to construction times. Such an assumption (in the context of travel within the already constructed parts of the network) was used in our previous work on network construction problems (Averbakh and Pereira 2012, 2015). The assumption that any vertex is always reachable from any other vertex without using the network under construction is realistic in a broader range of situations than the assumption that any point of the network is always reachable. For example, if the vertices represent population centers that need to be connected by major roads or railways (network under construction), there are often secondary local (perhaps unpaved) roads that connect them; if the network under construction represents an underground subway system whose tunnels are damaged as a result of an earthquake, its vertices (stations) are reachable from each other by on-the-ground roads/streets; etc. Finally, there definitely can be network construction settings where the model of this paper is unrealistic and not applicable, and a different class of models (e.g., those of Averbakh and Pereira 2012, 2015) should then be used.

Our focus is on developing exact solution approaches. The problem is strongly NP-hard on general networks, however we show that it is polynomially solvable on trees. For general networks, we present a mixed-integer linear programming (MILP) formulation, analyze structural properties of the problem, develop a number of lower bounds on the optimal objective value based on combinatorial structure of the problem, present a branch-and-bound algorithm that uses the developed lower bounds, and discuss results of computational experiments. The experiments indicate that the MILP formulation can be used only for very small instances, but the branch-and-bound algorithm can be used for medium-size instances. The branch-and-bound algorithm was also very successful when applied to instances based on the real-life data from 2010 Chilean earthquake.

2. Literature Review

Let us give a brief overview of the most directly related literature. Baxter et al. (2014) and Engel et al. (2017) consider incremental network design problems, which are defined as follows. Starting with an initial network, at each stage one new edge is installed, and it is required to minimize the total value over all stages of a network performance metric such as the shortest path length (Baxter et al. 2014) or the minimum spanning tree length (Engel et al. 2017). Connectivity issues are not present in these problems because in the initial network the vertices of interest are assumed to be already connected. Kalinowski et al. (2015) study incremental network design problems where it is required to maximize the cumulative maximum flow in the network over all stages. Interestingly, while problems of this type are typically NP-hard, in the case of the spanning tree based objective the incremental network design problem is polynomially solvable (Engel et al. 2017) because of its matroidal structure.

Guha et al. (1999) study the power outage recovery problem. When some relay vertices fail in an electric power network, connectivity between customer vertices and generator vertices may be destroyed and then needs to be restored. In the model considered in Guha et al. (1999), the failed vertices are repaired in stages where only a subset of failed vertices can be repaired in every stage because of limited budget. It is required to minimize the total weighted waiting time of disconnected customers. The focus of the work is on theoretical approximation results.

Nurre et al. (2012) introduce integrated network design and scheduling (INDS) problems where it is required to schedule installation of capacitated arcs in a network by a set of identical work crews that operate in parallel. The arcs are installed to increase the maximum flow that can go through the network, and it is required to maximize the cumulative weighted flow over a finite time horizon. The problem setting is more general than that in Kalinowski et al. (2015) as it allows

parallel installation of arcs. The results include an integer programming formulation, valid inequalities, and heuristic dispatching rules. Nurre and Sharkey (2014) study various INDS problems with objectives based on different network performance metrics such as the maximum flow value, the minimum cost flow value, shortest path lengths, and the minimum spanning tree length. All the considered problems are NP-hard (Nurre and Sharkey 2014). Heuristic algorithms based on dispatching rules are developed in Nurre and Sharkey (2014). This class of INDS problems is general and can model many environments and objectives. However, the INDS models of Nurre et al. (2012), Nurre and Sharkey (2014) are based on dynamics of improvement of overall network characteristics such as the minimum spanning tree length, the minimum cost flow value, the maximum flow value, and the shortest path length, and do not seem to be directly suitable for representing our problem's objective based on the lateness of individual connections with respect to their due dates.

Averbakh and Pereira (2012, 2015) study network construction problems where construction of a network is performed by a server (construction crew) that can travel only within the already constructed part of the network. Because of this assumption, at any instant the set of already constructed edges is connected, which is different from the setting of the present paper where at any time construction can be conducted at any point of the network. The objective considered in Averbakh and Pereira (2012) is to minimize the total weighted recovery time of all vertices, where the recovery time of a vertex is the time when the vertex becomes connected with the initial location of the server (depot), which is specified in advance and can be at any vertex of the network. The objectives considered in Averbakh and Pereira (2015) are to minimize the maximum lateness of the vertices and to minimize the number of tardy vertices with respect to given due dates for the recovery times of the vertices. The maximum lateness minimization problem of Averbakh and Pereira (2015) (Problem L) can be viewed as a special case of the problem studied in the present paper, as we will discuss in the next section. In contrast with our problem where connections between any pairs of vertices may be important, in problems from Averbakh and Pereira (2012, 2015) only the connections between vertices and the depot are important. The main contributions of Averbakh and Pereira (2012, 2015) are the obtained lower bounds on the objective function value based on analysis of combinatorial properties of the problems, which are incorporated in a branch-and-bound algorithm. Note that the structure of the branch-and-bound algorithms of Averbakh and Pereira (2012, 2015) is entirely different from the structure of the branch-and-bound algorithm presented in this paper. This will be discussed in detail in Section 8.

Averbakh (2012) show that a broad class of network construction problems with multiple servers and depots, which includes the multiserver/multidepot versions of the problems from Averbakh and Pereira (2012, 2015), is polynomially solvable on path networks. In Averbakh (2017), he shows that if the objective is to minimize the maximum recovery time of the vertices (the makespan), then multiserver/multidepot network construction problems where servers can travel only in the already constructed parts of the network are polynomially solvable on trees and cactus networks.

Additional references to papers that combine network design and scheduling aspects can be found in the literature review sections of Baxter et al. (2014), Nurre et al. (2012), Nurre and Sharkey (2014).

3. The Problem

Let $G = (V, E)$ be a connected network that has to be constructed with the set of vertices $V = \{1, \ldots, n\}$ and the set of undirected edges E, where $|E| = m$. For each edge $e \in E$, let $c_e > 0$ be the length of the edge. For any pair of vertices $i, j \in V$, a nonnegative due date d_{ij} is given, $d_{ij} = d_{ji}$, $d_{ii} = +\infty$ for any $i \in V$. A pair $i, j \in V$, $i \neq j$, such that $d_{ij} < +\infty$ will be called a *relevant pair*. We assume that there is at least one relevant pair. For any real numbers a, b, let $[a, b]$ denote the real interval with the endpoints a, b. For any integer numbers k, p, $k \leq p$, let $[k : p] = \{k, k+1, \ldots, p\}$. An edge with endpoints i, j will be denoted by $\{i; j\}$.

Construction starts at time 0. At any instant $t \geq 0$, construction can be performed at any point of the network. The construction speed is fixed and is equal to one unit of length per one unit of time. The instant when two vertices i, j become connected by an already constructed path is called the *connection time* for i, j and is denoted t_{ij}. (We assume $t_{ii} = 0$ for any $i \in V$.) For any pair $i, j \in V$, the value $t_{ij} - d_{ij}$ will be called (i, j)-*lateness*. Our purpose is to choose a construction schedule that minimizes the maximum lateness $\max_{i,j \in V}(t_{ij} - d_{ij})$. This problem will be called *Problem NCPC-L* (NCPC stands for network construction with pairwise connection). The optimal objective value for Problem NCPC-L will be denoted as Z^*.

As mentioned in the introduction, the assumption that at any instant construction can be performed at any point of the network can be interpreted as follows: the construction resources (workers, equipment, materials) at any time can be relocated from any point to any other point of the network, even if the two points are not connected by an already constructed path, by using some means of transportation that do not depend on the network under construction, with relocation times

that are negligible with respect to construction times. Since the objective function is monotonically nondecreasing in the connection times t_{ij} and the overall construction speed is fixed, the following two properties hold:

(a) Even if there is a possibility to perform construction at different places simultaneously, there is an optimal solution where at any time only one edge is being constructed. That is, it is not beneficial to split the limited resources between different edges.

(b) There is an optimal solution without preemption, where construction of an edge is not interrupted once it has been started.

To see this, for any solution A, consider solution B where the edges are constructed one at a time without preemption and idle times in the order of the completion times of the edges in solution A. Then clearly, completion times of all edges in solution B are not greater than the completion times of the edges in solution A, and if solution A is optimal, then solution B must be optimal as well. So, we limit our attention to schedules that satisfy these two properties. Thus, the construction can be considered to proceed in steps, where at each step one edge is constructed.

Remark 1. Note that a solution that satisfies properties a, b above can be implemented with transportation of construction resources only between the vertices of the network. Thus, we get an equivalent problem if instead of immediate accessibility of any point of the network, we only require that any vertex of the network can always be reached from any other vertex in negligible time by some modes of transportation that do not use the network under construction.

The edges that are constructed before all pairs $i, j \in V$ become connected are called *essential*. Observe that there is an optimal solution where the essential edges form a spanning tree of G, and therefore are constructed in the first $n-1$ steps. When the essential edges have been constructed, all pairs are connected, and the order of constructing other edges is irrelevant. Thus, *the problem is to choose a spanning tree, and an optimal order of constructing its edges, so as to minimize the considered objective function* $\max_{i,j\in V}(t_{ij} - d_{ij})$. This will be considered as the formal definition of the problem. In the remainder of the paper, by a *solution* to Problem NCPC-L we will understand a sequence of $n-1$ edges that form a spanning tree.

Observe that, in contrast with the network construction problems considered in Averbakh and Pereira (2012, 2015), in Problem NCPC-L for a spanning tree of essential edges *any* order of constructing its edges is feasible, since we assume that the resources can be transported without using the network under construction. So, Problem NCPC-L differs from the network construction problems of Averbakh and Pereira (2012, 2015) not only in the objective function but also in the set of feasible solutions. In problems from Averbakh and Pereira (2012, 2015), feasible solutions are sequences of edges that grow a spanning tree from the depot, that is, that at any time form a single cycle-free connected component that contains the depot. In Problem NCPC-L, feasible solutions are sequences of edges that are not limited to forming a single connected component at all times, but can at times form several nontrivial cycle-free connected components (eventually merging into a single spanning tree). In multidepot network construction problems of Averbakh (2012), a feasible solution cannot be represented by a sequence of edges at all, because in these problems different servers work at different parts of the network simultaneously, starting from their respective depots, and cannot move across parts of the network that have not been constructed yet.

Suppose a solution S is considered. A pair (i, j) of vertices from V is called a *critical pair for* S, if it has the largest (i, j)-lateness under S among all pairs (i, j), $i, j \in V$. For any $i, j \in V$, $i \neq j$, the edge whose construction in S results in connecting the pair (i, j) is called the *critical edge for* (i, j) *in* S.

Let us briefly discuss the relation between Problem NCPC-L and some other combinatorial optimization problems.

If there is only one relevant pair $i, j \in V$, then Problem NCPC-L is equivalent to the problem of finding a shortest path between i and j. Indeed, the objective function then will try to minimize the connection time for the pair i, j, which is clearly done by constructing a shortest path between i, j.

If all due dates d_{ij}, $i \neq j$, $i, j \in V$ are equal, then Problem NCPC-L is equivalent to the minimum spanning tree problem. Indeed, the objective function then will try to minimize the time when all vertices become connected, which is clearly done by constructing a minimum spanning tree.

If all relevant pairs have a common node i, then clearly there is an optimal solution where at any time all already constructed edges form a connected subnetwork of G that contains node i. Thus, in this case Problem NCPC-L is equivalent to Problem L considered in Averbakh and Pereira (2015) (described in Section 2) with a depot at node i, and Problem L of Averbakh and Pereira (2015) is equivalent to this special case of Problem NCPC-L (to model Problem L with depot at node i in terms of Problem NCPC-L, treat the due date of any vertex j as the due date for the pair i, j). So, mathematically, Problem L of Averbakh and Pereira (2015) is a special case of Problem NCPC-L, and Problem NCPC-L is a generalization of Problem L. Therefore, *Problem NCPC-L is strongly NP-hard on general networks* because its special case, Problem L considered in Averbakh and Pereira (2015), is strongly NP-hard.

4. MILP Formulation

To develop an MILP formulation, we will have several groups of variables. Boolean variables x_e^k, $e \in E$, $k \in [1:n-1]$, will define the essential edges and the order of their construction; $x_e^k = 1$ will mean that at Step k edge e is constructed. Auxiliary continuous variables z_{ij}^k, $k \in [1:n]$, $i, j \in V$, which will be restricted to take values only from the interval $[0,1]$, will have the following properties: if the nodes i, j are still disconnected when Step $k-1$ is finished, then z_{ij}^k will be forced to take value 0; otherwise, z_{ij}^k will be allowed to take value 1. Continuous variables q_k, $k \in [1:n-1]$, will represent the time needed to perform the first k steps, that is, to construct the first k essential edges. Continuous variables T_{ij}, $i, j \in V$, $i < j$, will represent the connection times t_{ij}.

For any edge $e \in E$, let a_e and b_e denote the two endpoints of e, $a_e < b_e$ (remember that a_e and b_e are integers from $[1:n]$ since $V = [1:n]$). To ensure the properties of variables z_{ij}^k mentioned above, we will use additional auxiliary continuous variables $\alpha_{i,e,j}^k$, $\beta_{i,e,j}^k$, $i, j \in V$, $i < j$, $e \in E$, $k \in [2:n]$, which will be restricted to take values only from $[0,1]$. These variables will have the following properties. Variable $\alpha_{i,e,j}^k$ will be forced to take value 0 unless the following three conditions are satisfied simultaneously:

(A1) Edge e has already been constructed when Step $k-1$ is finished.

(A2) Nodes i and a_e are connected by the beginning of Step $k-1$.

(A3) Nodes b_e and j are connected by the beginning of Step $k-1$.

If all three conditions A1–A3 are satisfied, $\alpha_{i,e,j}^k$ will be allowed to take any value in $[0,1]$.

Variable $\beta_{i,e,j}^k$ will be forced to take value 0 unless the following three conditions are satisfied simultaneously:

(B1) Edge e has already been constructed when Step $k-1$ is finished.

(B2) Nodes i and b_e are connected by the beginning of Step $k-1$.

(B3) Nodes a_e and j are connected by the beginning of Step $k-1$.

If all three conditions B1–B3 are satisfied, $\beta_{i,e,j}^k$ will be allowed to take any value in $[0,1]$.

Thus, variable z_{ij}^k for $k \in [2:n]$ should be allowed to take value 1 if and only if for some edge e, either $\alpha_{i,e,j}^k$ or $\beta_{i,e,j}^k$ is allowed to take value 1.

The formulation is as follows:

$$\text{minimize } \theta \tag{1}$$

subject to

$$\theta \geq T_{ij} - d_{ij}, \quad \text{for all } i, j \in V, i < j; \tag{2}$$

$$T_{ij} \geq q_k - M z_{ij}^k, \quad \text{for all } i, j \in V, i < j, k \in [1:n-1]; \tag{3}$$

$$q_k = \sum_{t=1}^{k} \sum_{e \in E} c_e x_e^t, \quad \text{for all } k \in [1:n-1]; \tag{4}$$

$$z_{ij}^1 = 0, \quad \text{for all } i, j \in V, i < j; \tag{5}$$

$$z_{ij}^n = 1, \quad \text{for all } i, j \in V; \tag{6}$$

$$z_{ii}^k = 1, \quad \text{for all } k \in [1:n-1], i \in V; \tag{7}$$

$$z_{ij}^k = z_{ji}^k, \quad \text{for all } i, j \in V, i < j, k \in [1:n-1]; \tag{8}$$

$$z_{ij}^k \geq z_{ij}^{k-1}, \quad \text{for all } k \in [2:n-1], i, j \in V, i < j; \tag{9}$$

$$x_e^k \in \{0,1\}, \quad \text{for all } e \in E, k \in [1:n-1]; \tag{10}$$

$$0 \leq z_{ij}^k \leq 1, \quad \text{for all } i, j \in V, k \in [1:n-1]; \tag{11}$$

$$\sum_{e \in E} x_e^k = 1, \quad \text{for all } k \in [1:n-1]; \tag{12}$$

$$\sum_{k \in [1:n-1]} x_e^k \leq 1, \quad \text{for all } e \in E; \tag{13}$$

$$0 \leq \alpha_{i,e,j}^k \leq 1, \quad 0 \leq \beta_{i,e,j}^k \leq 1, \quad \text{for all } i, j \in V, i < j, e \in E, k \in [2:n]; \tag{14}$$

$$\alpha_{i,e,j}^k \leq z_{ia_e}^{k-1}, \quad \alpha_{i,e,j}^k \leq \sum_{t=1}^{k-1} x_e^t, \quad \alpha_{i,e,j}^k \leq z_{b_e j}^{k-1}, \quad \text{for all } i, j \in V, i < j, e \in E, k \in [2:n]; \tag{15}$$

$$\beta_{i,e,j}^k \leq z_{ib_e}^{k-1}, \quad \beta_{i,e,j}^k \leq \sum_{t=1}^{k-1} x_e^t, \quad \beta_{i,e,j}^k \leq z_{a_e j}^{k-1}, \quad \text{for all } i, j \in V, i < j, \ e \in E, k \in [2:n]; \tag{16}$$

$$z_{ij}^k \leq \sum_{e \in E} (\alpha_{i,e,j}^k + \beta_{i,e,j}^k), \quad \text{for all } i, j \in V, i < j, k \in [2:n]. \tag{17}$$

Variables x_e^k, $e \in E$, $k \in [1:n-1]$ are Boolean. Variables θ; q_k, $k \in [1:n-1]$; z_{ij}^k, $i, j \in V$, $k \in [1:n]$; T_{ij}, $i, j \in V$, $i < j$; and $\alpha_{i,e,j}^k$, $\beta_{i,e,j}^k$, $i, j \in V$, $i < j$, $e \in E$, $k \in [2:n]$ are continuous. Value M is a sufficiently large constant, for example, $M = (n-1)c_{\max}$, where $c_{\max}$ is the length of the longest edge of G. This formulation has $(n-1)m$ Boolean variables, $O(n^3 m)$ continuous variables, and $O(n^3 m)$ constraints.

Theorem 1. (1)–(17) *is a correct formulation for Problem NCPC-L.*

Proof. Constraints (10), (12)–(13) ensure that variables x_e^k have the desired meaning. Constraints (4) ensure that variables q_k, $k \in [1:n-1]$ have the desired meaning. Now we need to prove the following statement.

Main statement. Variables z_{ij}^k for all $k \in [1:n]$, $i, j \in V$ have the desired properties. That is, given a fixed assignment of 0–1 values to variables x_e^k that satisfies constraints (12) and (13) and therefore defines a set of $n-1$ essential edges and an order of their construction, each variable z_{ij}^k is forced to take value 0 if after Step $k-1$ nodes i, j are still disconnected, and can take value 1 otherwise, that is, there is a feasible assignment of values to all variables $z_{ij}^k, \alpha_{i,e,j}^k, \beta_{i,e,j}^k$ where $z_{ij}^k = 1$ if after Step $k-1$ nodes i, j are connected by an already constructed path.

To prove the main statement, we use induction in k. Values z_{ij}^k for $k = 1$ are defined by the constraints (5), (7), (8), and they have the desired properties. Thus, we can say that the main statement holds for $k = 1$.

Induction hypothesis. Suppose that we have already proved that for all $i, j \in V$ variables z_{ij}^k have the desired properties for all $k \in [1: p]$ for some $p \in [1 : n-1]$, that is, the main statement holds for all $k \in [1: p]$. We need to prove that it also holds for $k = p+1$.

For any $i, j \in V$, $i < j$, consider the value z_{ij}^{p+1}. Suppose that after Step p, the nodes i, j are connected by at least one already constructed simple path P. Let $e = \{a_e; b_e\}$, $a_e < b_e$ be the edge of P that was constructed last, and suppose that in the path P the node a_e is closer to i than the node b_e. Then, conditions A1–A3 are satisfied, and according to the induction hypothesis $z_{ia_e}^p$ and $z_{b_e j}^p$ can take value 1. Observe also that $\sum_{t=1}^p x_e^t = 1$. Therefore, value $\alpha_{i,e,j}^{p+1}$ can take value 1 (see (15)), and thus z_{ij}^{p+1} can take value 1 (see (17)). If in the path P the node b_e is closer to i than the node a_e, the argument is similar, with value $\beta_{i,e,j}^{p+1}$ instead of $\alpha_{i,e,j}^{p+1}$.

Suppose now that the nodes i, j are not connected after Step p. Then, for any $e \in E$, at least one of the conditions A1–A3 and at least one of the conditions B1–B3 are not satisfied, so $\alpha_{i,e,j}^{p+1} = 0$ and $\beta_{i,e,j}^{p+1} = 0$ according to the constraints (15) and (16) and the induction hypothesis. Then, z_{ij}^{p+1} is forced to take value 0 by the constraints (17). Taking into account (8) and (7), we conclude that variables z_{ij}^{p+1}, $i, j \in V$ have the desired properties. This completes the proof of the main statement.

Given the main statement, constraints (6) ensure that after Step $n-1$, the constructed edges form a spanning tree. Also, because of (1)–(3), there is an optimal solution where each z_{ij}^k takes value 1 whenever it is allowed by other constraints, that is, (from the main statement) when i and j are connected after Step $k-1$. Note that for such a solution, $t_{ij} = \max_{k\in[1:n-1]}(q_k - Mz_{ij}^k)$. Constraints (3) ensure that $T_{ij} \geq t_{ij}$; in fact, taking into account (1) and (2), clearly there is an optimal solution where $T_{ij} = t_{ij}$. (1) and (2) define the objective. □

Note that constraints (9) are unnecessary, and are included just as additional cuts.

Remark 2. For scheduling problems, time-indexed formulations where time is discretized and Boolean variables represent decisions to start specific jobs at specific time instants, often are effective as they often produce stronger linear programming relaxations at the expense of having more variables (see, e.g., Sousa and Wolsey 1992). We also developed a time-indexed MILP formulation, but computational experiments indicated that it is inferior to the formulation above, so we do not report it.

5. Problem NCPC-L on Cycle-Free Networks

In this section, we assume that G is a tree. For any $i, j \in V$, let $P_{i,j}$ denote the only simple path between i and j. For any edge e, define $\tilde{d}(e)$ as the smallest of values d_{ij} such that $e \in P_{i,j}$. Values $\tilde{d}(e)$, $e \in E$ will be called *derived edge-dates*.

Theorem 2. *Constructing the edges of the tree G in the order of nondecreasing derived edge-dates is optimal for Problem NCPC-L*

Corollary 1. *Problem NCPC-L on a tree can be solved in $O(n \log n)$ time with $O(\hat{r}n)$ preprocessing, where $\hat{r}$ is the number of relevant pairs.*

Proof of Corollary 1. All $n-1$ derived edge-dates can be computed in $O(\hat{r}n)$ total time. Then, it takes $O(n \log n)$ time to sort them. □

Proof of Theorem 2. For an instance T of Problem NCPC-L on a tree G with due dates d_{ij}, define an instance T' on the same tree with due dates d'_{ij} as follows. For any pair of adjacent vertices i, j, $d'_{ij} = \tilde{d}(e)$, where $e = \{i; j\}$; for nonadjacent i, j, $d'_{ij} = +\infty$. Since for the instance T', only pairs of adjacent vertices may be relevant, and G is a tree, then when construction of an edge e is finished at most one relevant pair gets connected and the due date of this pair is $\tilde{d}(e)$. Therefore, an optimal solution for T' is obtained by constructing the edges according to nondecreasing values $\tilde{d}(e)$. To complete the proof, it is sufficient to show that instances T and T' have the same sets of optimal solutions and the same optimal objective values.

For any solution S (a sequence of edges of G), let $Z_T(S)$ and $Z_{T'}(S)$ be the objective values for S in T and T', respectively. Suppose that (i, j) is a critical pair for S in T and $e = \{a; b\}$ is the critical edge for (i, j) in S. Then, d_{ij} is the smallest of all d_{pq} such that $e \in P_{p,q}$; that is, $d_{ij} = \tilde{d}(e) = d'_{ab}$. Now, observe that under S, (i, j)-lateness in T is equal to (a, b)-lateness in T', and (a, b) is a critical pair for S in T' (otherwise (i, j) would not be a critical pair for S in T). Consequently, $Z_T(S) = Z_{T'}(S)$. The theorem is proven. □

6. Induced Due Dates

In the remainder of the paper, we assume that G is a general network.

Lemma 1. *For any $k \in [3: n]$ and any k distinct vertices $v_1, \ldots, v_k$, if $d_{v_1 v_k} > \max_{i\in[1:k-1]} d_{v_i v_{i+1}}$, then (v_1, v_k) cannot be a critical pair for any solution S.*

Proof. It is sufficient to observe that at any time before the instant when v_1 and v_k become connected, at least one of the $k-1$ pairs (v_i, v_{i+1}), $i \in [1: k-1]$ is not connected. Consequently, for any solution S, $t_{v_1 v_k} \leq t_{v_i v_{i+1}}$ for some $i \in [1: k-1]$, and thus using the lemma's condition $t_{v_1 v_k} - d_{v_1 v_k} < t_{v_i v_{i+1}} - d_{v_i v_{i+1}}$. □

Definition 1. For any $i, j \in V$, $i \neq j$, define

$$\hat{d}_{ij} = \min\Big\{d_{ij}, \min_{k, v_1, \ldots, v_k} \max\{d_{iv_1}, d_{v_1v_2}, d_{v_2v_3}, \ldots, d_{v_{k-1}v_k}, d_{v_kj}\}\Big\},$$

where the internal minimization is over all $k \in [1 : n-2]$ and all sequences $(v_1, v_2, \ldots, v_k)$ of k distinct vertices from $V \backslash \{i, j\}$. Values $\hat{d}_{ij}$, $i, j \in V$, $i \neq j$, will be called the *induced due dates*.

Theorem 3. *Replacing the original due dates d_{ij} with the induced due dates $\hat{d}_{ij}$ results in an equivalent instance of Problem NCPC-L. Specifically, it does not change the objective value of any solution S.*

Proof. The result follows from Lemma 1. □

A reason to use the induced due dates instead of the original due dates may be to reduce the range of the due dates involved. Induced due dates can make some of the lower bounds stronger (e.g., the bound $LB3$ from Section 7.1). They will also be useful in some proofs.

The following algorithm finds the induced due dates.

Algorithm 1 (Find induced due dates)

```
1: for c ∈ V do
2:    for a, b ∈ V\{c}, a < b do
3:       if d_ab > max{d_ac, d_cb} then
4:          d_ab := max{d_ac, d_cb}
5:       end if
6:       d_ba := d_ab
7:    end for
8: end for.
```

Observation 1. Algorithm 1 correctly finds the induced due dates in $O(n^3)$ time.

Proof. The order of complexity is straightforward. Correctness is proven using the standard arguments for the Floyd-Warshall algorithm. Algorithm 1 is a minor modification of the Floyd-Warshall algorithm (see, e.g., Ahuja et al. 1993, Chap. 5).

Let D_{ind} denote the set of (distinct) induced due dates.

7. Lower Bounds

In this section, we develop several lower bounds on the optimal objective value Z^*.

7.1. Lower Bounds $LB1$, $LB2'$, and $LB3$ for Z^*

Suppose that $V_1, \ldots, V_k$ are some pairwise disjoint nontrivial subsets of V. (A subset is called nontrivial if it contains more than one element.) Define the following *Shortest Set Connection problem*, which we will call Problem SSC$(V_1, \ldots, V_k)$.

Problem SSC$(V_1, \ldots, V_k)$. Find a subnetwork G' of G of minimum length so that for each $i \in [1 : k]$ all vertices of V_i are connected with each other in G'.

Observe that an optimal solution to Problem SSC$(V_1, \ldots, V_k)$ is a forest (a collection of subtrees) with at most k connected components. Let $F(V_1, \ldots, V_k)$ be the optimum objective value for Problem SSC$(V_1, \ldots, V_k)$. Observe that $F(V)$ is the length of the minimum spanning tree for G.

For any real value d, define graph $Q(d) = (V, \hat{E}(d))$, where for any $i, j \in V$, $i \neq j$, edge $\{i; j\}$ is present if and only if $d_{ij} \leq d$, that is, $\hat{E}(d) = \{\{i; j\} \mid d_{ij} \leq d\}$. Clearly, $Q(+\infty)$ is a complete graph, and for a negative d, $Q(d)$ is a graph without edges. Note that the edge set $\hat{E}(d)$ is not derived from the edge set E (some edges of $\hat{E}(d)$ may not be present in E) but is defined only by the due dates d_{ij} and the value d. Let $F^{(d)} = F(V_1, \ldots, V_k)$, where $V_1, \ldots, V_k$ are the nontrivial connected components of $Q(d)$. Let D denote the set of (distinct) finite due dates.

Lemma 2. *For any $d \in D$, $F^{(d)} - d$ is a lower bound for Z^*.*

Proof. Consider a $d \in D$. For any vertices i, j, $i \neq j$ from the same connected component of $Q(d)$, the induced due date $\hat{d}_{ij}$ is not greater than d. Also, for any solution, there is a pair of vertices (i, j), $i \neq j$ from the same connected component of $Q(d)$ such that $t_{ij} \geq F^{(d)}$. Then, for this pair (i, j), $t_{ij} - \hat{d}_{ij} \geq F^{(d)} - d$. Therefore, $Z^* \geq F^{(d)} - d$. □

Let $d^*_{\max}$ be the smallest value $d \in D$ such that $Q(d)$ is connected. Then, $F^{(d^*_{\max})} = F(V)$ is the length of a minimum spanning tree for G. Define

$$LB1 = F(V) - d^*_{\max}.$$

Value $LB1$ is easily computable and is a lower bound for Z^* according to Lemma 2.

Let us sort the values $d \in D$ in increasing order, $D = \{d_1 < d_2 < \cdots < d_r\}$. Value $d = d_i$ is called *influential* if graph $Q(d_i)$ has different nontrivial connected components than $Q(d_{i-1})$ (value d_1 is always influential). Clearly, there are at most $n - 1$ influential values d (because each influential value d corresponds to reducing the number of connected components of $Q(d)$ (trivial and nontrivial) at least by one); let D^* be the set of influential values d, $|D^*| \leq n - 1$. Note that $d^*_{\max} = \max\{d \mid d \in D^*\}$. Define $LB2(d) = F^{(d)} - d$ and

$$LB2 = \max\{LB2(d) \mid d \in D^* \backslash \{d^*_{\max}\}\}.$$

According to Lemma 2, $LB2$ is a lower bound for Z^*. Observe that $LB1 = LB2(d^*_{\max})$; that is why we exclude $d^*_{\max}$ from D^* in the definition of $LB2$. We treat $LB1$ separately because $LB1 = LB2(d^*_{\max})$ is easy to compute, in contrast with $LB2(d)$ for $d \in D^* \backslash \{d^*_{\max}\}$.

The best lower bound that can be obtained using Lemma 2 is $\max\{(F^{(d)} - d) \mid d \in D\}$, which seems to require computing $F^{(d)}$ for $O(n^2)$ values $d \in D$. The following result demonstrates that it is sufficient to use only at most $n - 1$ values $d \in D^*$ required to compute $LB1$ and $LB2$.

Theorem 4. $\max\{(F^{(d)} - d) \mid d \in D\} = \max\{LB1, LB2\}$.

Proof. If $d \in D$ is not influential, then there is an influential d', $d' < d$, such that $Q(d)$ and $Q(d')$ have the same nontrivial connected components $V_1, \ldots, V_k$. Thus, $F^{(d)} = F^{(d')}$, and $F^{(d)} - d < F^{(d')} - d'$. So, $F^{(d)} - d < \max\{LB1, LB2\}$, and d need not to be considered. □

Although to compute $LB2$ it is sufficient to solve at most $n-2$ instances of Problem SSC$(V_1, \ldots, V_k)$, the bound $LB2$ is impractical because Problem SSC$(V_1, \ldots, V_k)$ is strongly NP-hard (since the strongly NP-hard Steiner tree problem (Garey and Johnson 1979) is a special case), and thus obtaining value $F^{(d)}$ is in general strongly NP-hard. So, instead of solving Problem SSC$(V_1, \ldots, V_k)$ exactly, it makes sense to use some efficiently computable lower bounds. Suppose some lower bounds $F'^{(d)} \le F^{(d)}$ are known for values $F^{(d)}$, $d \in D^* \backslash \{d^*_{\max}\}$; define $LB2'(d) = F'^{(d)} - d$ and

$$LB2' = \max\ \{LB2'(d) \mid d \in D^* \backslash \{d^*_{\max}\}\}.$$

Then, $LB2'$ is a valid lower bound for Z^*.

The following result clarifies the connection between the influential and the induced due dates.

Theorem 5. $D^* = D_{\text{ind}}$.

Proof. First, we prove that if a due date $d \in D$ is not influential, then $d \notin D_{\text{ind}}$. Indeed, suppose that for $d \in D$ there is $d' \in D$, $d' < d$, such that $Q(d)$ and $Q(d')$ have the same connected components. Then, for any vertices a, b from the same connected component of $Q(d)$, the induced due date $\hat{d}_{ab}$ is not greater than d' because there are $i_1, i_2, \ldots, i_t$ such that d_{ai_1}, $d_{i_1 i_2}, \ldots, d_{i_{t-1} i_t}$, $d_{i_t b}$ are all not greater than d'. Therefore, $\hat{d}_{ab} \ne d$. Also, for any vertices a, b that belong to different connected components of $Q(d)$, $\hat{d}_{ab} > d$. Thus, there is no a, b such that $\hat{d}_{ab} = d$.

Now, we prove that if a due date $d \in D$ is influential, then $d \in D_{\text{ind}}$. Suppose that $d \in D$ is influential; then there are i, j, $i \ne j$, such that $d = d_{ij}$, and for any $d' < d$, i and j belong to different connected components of $Q(d')$. Then, $\hat{d}_{ij}$ cannot be less than d, and clearly $\hat{d}_{ij} \le d_{ij} = d$. Thus, $\hat{d}_{ij} = d$. □

Corollary 2. $|D_{\text{ind}}| \le n - 1$.

Let l_{ij} denote the shortest distance in G between vertices i, j. Define

$$LB3 = \max\{l_{ij} - d_{ij} \mid i, j \in V,\ i < j\}.$$

Observation 2. $LB3$ is a valid lower bound for Z^*.

Proof. It is sufficient to observe that l_{ij} is a lower bound for t_{ij}. □

Let us consider a small example. Consider a network with four nodes a, b, c, d, and edges $\{a; b\}, \{b; c\}, \{c; d\}$ of the same length 1. Suppose that the relevant due dates are $d_{ab} = d_{cd} = 1$ and $d_{bc} = 10$. Then $LB1 = -7$, $LB2 = 1$ (because $LB2(1) = 1$), and $LB3 = 0$, so $LB2$ is the dominant bound. If we change d_{bc} and make it 1.5 instead of 10, then $LB2$ and $LB3$ do not change but $LB1$ becomes 1.5 and is now the dominant bound.

Remark 3. Note that always $LB3 \le \max\{LB1, LB2\}$. To see this, suppose that $LB3 = l_{ij} - d_{ij}$ for some $i, j \in V$, then $F^{(d_{ij})} \ge l_{ij}$ and therefore $F^{(d_{ij})} - d_{ij} \ge l_{ij} - d_{ij}$. Now we can use Theorem 4. However, since $LB2$ is difficult to compute and we will use different versions of lower bound approximation $LB2'$ instead of $LB2$, it is possible that $LB3$ can be strictly dominant (i.e., stronger than other used bounds) in some instances.

7.2. Lower Bounds for $F(V_1, \ldots, V_k)$ and Lower Bounds $LB4$ and $LB5(d)$ for Z^*

The lower bound $LB2'$ is based on having some lower bound $F'^{(d)}$ for the value $F^{(d)} = F(V_1, \ldots, V_k)$, where $V_1, \ldots, V_k$ are the connected nontrivial components of $Q(d)$. Thus, the strength of the lower bound $LB2'$ depends on the strength of available lower bounds for the optimal objective value $F(V_1, \ldots, V_k)$ of Problem SSC$(V_1, \ldots, V_k)$. In this subsection, we present some lower bounds for $F(V_1, \ldots, V_k)$, where $V_1, \ldots, V_k$ are arbitrary pairwise disjoint nontrivial subsets of V. These bounds can be used in the context of $LB2'$ to obtain lower bounds for Z^*.

7.2.1. Bounds $B_0(V_1, \ldots, V_k)$ and $LB4$.

For a vertex $v \in V$, define $c_{\min}(v) = \min\{c_e \mid e \text{ is incident to } v\}$. For a subset $V' \subset V$, define $c_{\min}(V') = \min\{c_{\min}(v) \mid v \in V'\}$. Define

$$B'_0(V_1, \ldots, V_k) = \sum_{i=1}^{k} \sum_{v \in V_i} c_{\min}(v) - \sum_{i=1}^{k} c_{\min}(V_i).$$

Observation 3. $B'_0(V_1, \ldots, V_k) \le F(V_1, \ldots, V_k)$.

Proof. An optimal solution to Problem SSC$(V_1, \ldots, V_k)$ consists of at most k vertex-disjoint subtrees, which together include all vertices from $V_1 \cup V_2 \cup \cdots \cup V_k$, and such that for each $i \in [1:k]$ all vertices from V_i belong to only one of these subtrees. Each such subtree can be considered as rooted at its vertex that minimizes $c_{\min}(v)$ over the vertices of one of V_i that is spanned by the subtree, and then we can define a one-to-one correspondence between the nonroot vertices of the subtree and its edges so that each edge of the subtree is assigned to one of its endpoints. The inequality follows. The idea of this bound is motivated by the lower bound on the Steiner tree value presented in Shore et al. (1982). □

Consider the network $\tilde{G}(V_1,\dots,V_k)=(\tilde{V},\tilde{E})$, which is obtained from G by deleting all edges that are not incident to any vertex from $V_1\cup\cdots\cup V_k$ and all vertices that are not adjacent to any vertex from $V_1\cup\cdots\cup V_k$. In other words, $\tilde{G}(V_1,\cdots,V_k)$ consists of all vertices from $V_1\cup\cdots\cup V_k$ and those adjacent to them, and all edges from E that have at least one endpoint in $V_1\cup\cdots\cup V_k$. A (*minimum*) *spanning subnetwork* for $\tilde{G}(V_1,\dots,V_k)$ is a (shortest) cycle-free subnetwork of $\tilde{G}(V_1,\dots,V_k)$ that spans all vertices of $\tilde{G}(V_1,\dots,V_k)$ and has the same number of connected components as $\tilde{G}(V_1,\dots,V_k)$; it is a (minimum) spanning tree of $\tilde{G}(V_1,\dots,V_k)$ if $\tilde{G}(V_1,\dots,V_k)$ is connected, and is a collection of (minimum) spanning trees for the connected components of $\tilde{G}(V_1,\dots,V_k)$ otherwise. Let $e_1,\dots,e_{\tilde{n}}$ be the edges of a minimum spanning subnetwork of $\tilde{G}(V_1,\dots,V_k)$ sorted in the order of nondecreasing lengths. Since the set $\{e_1,\dots,e_{\tilde{n}}\}$ is cycle-free, it can be obtained by the standard greedy algorithm.

Let $r(V_1,\dots,V_k)=\sum_{i=1}^{k}|V_i|-k$; note that $r(V_1,\dots,V_k)\le\tilde{n}$. Define

$$B_0''(V_1,\dots,V_k)=\sum_{j=1}^{r(V_1,\dots,V_k)}c_{e_j}.$$

Observation 4. *$B_0''(V_1,\dots,V_k)\le F(V_1,\dots,V_k)$.*

Proof. An optimal solution to Problem SSC$(V_1,\dots,V_k)$ is a forest that consists of at most k vertex-disjoint subtrees of G. This forest has at least $r(V_1,\dots,V_k)$ edges in $\tilde{G}(V_1,\dots,V_k)$. Since cycle-free sets of edges of $\tilde{G}(V_1,\dots,V_k)$ form a graphic matroid, $B_0''(V_1,\dots,V_k)$ is a lower bound on the total length of any cycle-free set of $r(V_1,\dots,V_k)$ edges of $\tilde{G}(V_1,\dots,V_k)$. □

Thus, $B_0'(V_1,\dots,V_k)$ and $B_0''(V_1,\dots,V_k)$ are valid lower bounds for $F(V_1,\dots,V_k)$. These bounds are simple and very easy to compute.

Now we combine the ideas of the bounds $B_0'(V_1,\dots,V_k)$ and $B_0''(V_1,\dots,V_k)$ to obtain a better bound. For any $i\in[1:k]$, let V_i' be the set obtained from V_i by deleting one vertex with the smallest value $c_{\min}(v)$; thus, $|V_i'|=|V_i|-1$, $\sum_{i=1}^{k}|V_i'|=r(V_1,\dots,V_k)$. Sort the $r(V_1,\dots,V_k)$ values $c_{\min}(v)$, $v\in\bigcup_{i=1}^{k}V_i'$ in nonincreasing order (from largest to smallest); let $\Omega=(q_1,q_2,\dots,q_{r(V_1,\dots,V_k)})$ be the obtained list. For any $t\in[0:r(V_1,\dots,V_k)]$ define

$$B_0^{(t)}(V_1,\dots,V_k)=\sum_{i=1}^{t}q_i+\sum_{j=1}^{r(V_1,\dots,V_k)-t}c_{e_j}.$$

Notice that $B_0^{(0)}(V_1,\dots,V_k)=B_0''(V_1,\dots,V_k)$, $B_0^{(r(V_1,\dots,V_k))}=B_0'(V_1,\dots,V_k)$.

Lemma 3. *For any $t\in[0:r(V_1,\dots,V_k)]$, $B_0^{(t)}(V_1,\dots,V_k)$ is a valid lower bound for $F(V_1,\dots,V_k)$.*

Proof. Value $\sum_{i=1}^{t}q_i$ is a lower bound on the total length of some t edges of the subtrees defined in the proof of Observation 3 (the edges that correspond to vertices that define values q_i, $i\in[1:t]$, using the correspondence between edges and vertices mentioned in the proof of Observation 3). Note that these t edges belong to $\tilde{G}(V_1,\dots,V_k)$. Then, $\sum_{j=1}^{r(V_1,\dots,V_k)-t}c_{e_j}$ is a lower bound on the total length of any $r(V_1,\dots,V_k)-t$ other edges of these subtrees that also belong to $\tilde{G}(V_1,\dots,V_k)$, because this is a lower bound on the total length of any $r(V_1,\dots,V_k)-t$ edges of $\tilde{G}(V_1,\dots,V_k)$ that do not create cycles and taking into account that in total the subtrees have at least $r(V_1,\dots,V_k)$ edges from $\tilde{G}(V_1,\dots,V_k)$. □

Define

$$B_0(V_1,\dots,V_k)=\max_{t\in[0:r(V_1,\dots,V_k)]}B_0^{(t)}.$$

Then, $B_0(V_1,\dots,V_k)$ is a valid lower bound for $F(V_1,\dots,V_k)$, which improves upon $B_0'(V_1,\dots,V_k)$ and $B_0''(V_1,\dots,V_k)$.

For any $d\in D^*\backslash\{d^*_{\max}\}$, let $LB4(d)$ be the bound $LB2'(d)$ where $B_0(V_1,\dots,V_k)$ is used as a lower bound $F'^{(d)}$ for $F^{(d)}$, and let $LB4=\max\{LB4(d)\mid d\in D^*\backslash\{d^*_{\max}\}\}$.

Remark 4. Note that the whole network G can be used instead of $\tilde{G}(V_1,\dots,V_k)$ for computing $B_0''(V_1,\dots,V_k)$ and $B_0^{(t)}(V_1,\dots,V_k)$, that is, the edges of a minimum spanning tree for G in the order of nondecreasing lengths can be used as $e_1,e_2,\dots$. This would avoid computing and keeping the minimum spanning subnetworks for different $\tilde{G}(V_1,\dots,V_k)$, thus somewhat simplifying the computations, but may result in weaker bounds.

7.2.2. Lagrangean Relaxation Bound $B_B(V_1,\dots,V_k)$. Let $T=\bigcup_{i=1}^{k}V_i$, $\bar{T}=V\backslash T$. In Beasley (1989), the author presented a lower bound for the minimum Steiner tree value. Now we present a bound that is an extension of the Beasley bound to the case of Problem SSC$(V_1,\dots,V_k)$. To define our bound, let us first obtain an auxiliary network $G^0=(V^0,E^0)$ by modifying the network G as follows:

1. Add a new vertex 0 to G.
2. Select arbitrarily a vertex $v(i)$ in each V_i, $i\in[1:k]$.
3. Add edges $\{0;v(i)\}$, $i\in[1:k]$, of length 0, and edges $\{0;v\}$, $v\in\bar{T}$, also of length 0.

Now, we define the following auxiliary problem.

Problem RELAX. Find a minimum spanning tree for G^0 subject to the additional restriction that in this spanning tree any vertex $i\in\bar{T}$ that is connected to vertex 0 by the edge $\{0;i\}$ must have degree 1.

Let Z^R denote the optimum objective value for Problem RELAX.

Lemma 4. *$Z^R\le F(V_1,\dots,V_k)$.*

Proof. An optimal solution X^* for Problem SSC($V_1, \ldots, V_k$) can be extended to a feasible solution for Problem RELAX with the same objective value as follows: X^* consists of at most k vertex-disjoint subtrees of G that span together all vertices in T, such that for any $i \in [1:k]$ all vertices from V_i belong to only one of these subtrees. For any such subtree from X^*, choose an $i \in [1:k]$ such that $v(i)$ belongs to this subtree, and add edge $\{0; v(i)\}$ to the solution. Then, for each vertex $j \in \bar{T}$ that is not spanned by X^*, add edge $\{0; j\}$ to the solution. Clearly, we obtain a spanning tree for G^0, of the same total length as X^* since all added edges have zero length. □

For any $i \in \bar{T}$, let $P_i = \{j \in V \mid \{i; j\} \in E\}$. Introducing variables x_e, $e \in E^0$, where

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is included in the solution,} \\ 0 & \text{otherwise,} \end{cases}$$

we can formulate Problem RELAX as follows:

$$\text{Minimize} \sum_{e \in E^0} c_e x_e,$$

s.t.

$$\begin{aligned} &\{e,\ x_e = 1\} \text{ is a spanning tree for } G^0; \\ &x_{\{0;i\}} + x_{\{i;j\}} \le 1, \quad \text{for all } i \in \bar{T},\ j \in P_i; \\ &x_e \in \{0,1\}, \quad \text{for all } e \in E^0. \end{aligned} \tag{18}$$

Inequalities (18) ensure that if edge $\{0; i\}$ is used, $i \in \bar{T}$, then no other edge adjacent to i is used. Relaxing constraints (18) in a Lagrangean manner with Lagrange multipliers λ results in a minimum spanning tree subproblem on G^0 with modified edge lengths, and its solution produces a lower bound for Z^R; let us call this bound $Z^{LR}(\lambda)$. Then, λ can be optimized iteratively by standard subgradient optimization (Beasley 1989), to strengthen the bound $Z^{LR}(\lambda)$. Let $B_B(V_1, \ldots, V_k)$ be the bound obtained after such optimization; this is the Lagrangean relaxation bound we use.

Note that if $k = 1$, then $B_B(V_1, \ldots, V_k)$ is the lower bound proposed in Beasley (1989) for the minimum Steiner tree problem, as Problem SSC(V_1) is the minimum Steiner tree problem. Note also that if $k = 1$, then the inequality in Lemma 4 holds as equality. If $k > 1$, then the inequality in Lemma 4 can be strict. Problem RELAX can be interpreted as the problem of finding at most k vertex-disjoint subtrees of G of minimum total length that span together all vertices in T, so that each of these subtrees contains at least one vertex from $\{v(i),\ i \in [1:k]\}$. For $k > 1$, the choice of vertices $v(i)$ in the sets V_i can influence the strength of the bound, but we simply choose the lowest-numbered vertex from V_i as $v(i)$ since the Lagrangean bound is already time-consuming.

For any $d \in D^* \backslash \{d^*_{\max}\}$, let $LB5(d)$ be the bound $LB2'(d)$ where $B_B(V_1, \ldots, V_k)$ is used as a lower bound $F'^{(d)}$ for $F^{(d)}$.

8. Branch-and-Bound Algorithm

8.1. Initial Solution and the General Structure of the Algorithm

An initial solution is obtained before branching using the following heuristic approach. Since Problem NCPC-L is polynomially solvable on trees, first we obtain a minimum spanning tree for G and solve Problem NCPC-L on the tree. This will be called *the minimum spanning tree heuristic*. The minimum spanning tree heuristic is combined with a local search; a neighbor of a spanning tree is obtained by adding an edge to the tree and deleting an edge from the resulting cycle. (Alternatively, an edge can be deleted, and another edge added from the resulting cut to restore connectivity, which defines the same neighborhood.) For each neighbor (a new spanning tree), Problem NCPC-L is solved on this spanning tree to obtain the optimal order of constructing its edges and the corresponding objective function value; if there is an improvement, the current spanning tree is updated, and so on. The heuristic stops when no objective function improvement is found within the current neighborhood. This will be called *the spanning tree with local search heuristic*.

For terminological clarity, the term "nodes" will be used for the nodes of the search tree in branch and bound, and the term "vertices" will be used for vertices of the network. In our branch-and-bound scheme, *branching is on inclusion/exclusion of a particular edge in/from the set of essential edges* (without specifying its position in the sequence). A *partial solution* that corresponds to a node of the search tree is specified by a set of *plus-edges* and a set of *minus-edges*. The plus-edges (minus-edges) are the edges that must be included in (excluded from) the solution as a result of the already made branching decisions, and possibly some additional analysis that is described below. The set of plus-edges is cycle-free. The additional analysis consists of implementing the following two rules.

Rule 1. If an edge e creates a cycle with the set of plus-edges, then e should be added to the set of minus-edges.

An edge of a connected network is called a *bridge* if its deletion would disconnect the network.

Rule 2. Let G'' be the network obtained from G by deleting all current minus-edges. If an edge e is a bridge in G'', e should be added to the set of plus-edges.

Rules 1 and 2 are applied to any new node of the search tree. Note that Rule 2 ensures that G'' is always connected.

The edges of G that are not included in the sets of plus-edges and minus-edges are the *candidate edges* for branching in this branching scheme. If the set of plus-edges has size $n - 1$, then a spanning tree of essential edges has already been obtained, and the exact value of the corresponding node of the search tree is obtained using the algorithm for trees from Section 4 that finds the optimal order of constructing the essential edges.

Remark 5. We note that the structure of our branch-and-bound scheme is entirely different from that in Averbakh and Pereira (2012, 2015), and results in an entirely different search tree. The branch-and-bound algorithms of Averbakh and Pereira (2012, 2015) grow the sequence of the already constructed edges, and branching is on the choice of the next edge to construct; thus, in Averbakh and Pereira (2012, 2015), branching is both on network design and sequencing decisions. Our algorithm does not grow a *sequence* of constructed edges, only a *set* of plus-edges (essential edges) without making decisions about the order of construction until a full set of essential edges (a spanning tree) is obtained. Thus, our branching is on a network design decision but not on a sequencing decision. A node of the search tree in Averbakh and Pereira (2012, 2015) represents an initial subsequence of edges to be built, with a fixed order. A node of the search tree in the current paper represents a partial decision on which edges to build/not build (i.e., to include/not include in the set of essential edges), without specifying the order and even without demanding that the edges assigned to be constructed (plus-edges) are the first to be constructed. Initially, we tried to use the branching logic similar to that of Averbakh and Pereira (2012, 2015), but it turned out to be far less effective than the branching scheme of our algorithm. One of the reasons appears to be that for Problem NCPC-L, if we grow the *sequence* of already constructed edges, there are many more potential choices of the next edge to construct (because the set of constructed edges does not have to be connected) than in the problems from Averbakh and Pereira (2012, 2015), especially in the initial stages of the algorithm, which results in a much larger search tree. Also, in the problems from Averbakh and Pereira (2012, 2015), a partial solution could be represented as a sequence of already recovered *vertices*, which allowed to further shorten the search space.

8.2. Implementation of the Lower Bounds for Partial Solutions

Now we discuss how the lower bounds developed in Section 7 should be adjusted to be used at nodes of the search tree in the branch-and-bound algorithm.

As discussed in the previous subsection, a partial solution that corresponds to a node of a search tree is defined by specifying plus-edges and minus-edges. A (shortest) spanning tree for G that contains all plus-edges and no minus-edges will be called a (*minimum*) *conditional spanning tree*. A minimum conditional spanning tree can be obtained with a minor adjustment of the Kruskal's algorithm (Ahuja et al. 1993, Chap. 13). To compute $LB1$ for a partial solution, the minimum spanning tree used in the bound is replaced with a minimum conditional spanning tree, and the formula for the bound remains the same.

To compute $LB3$ for a partial solution, the shortest distances between vertices are computed in the modified network G'', which is obtained from G by deleting all minus-edges. These distances are used as values l_{ij}, and the formula for $LB3$ remains the same. (Note that plus-edges are treated here as regular edges.)

To define $LB2$ for a partial solution, Problem SSC($V_1, \ldots, V_k$) that corresponds to $F^{(d)}$ is considered with the following additional constraint: G' must be extendable into a conditional spanning tree for G. This problem will be called the *conditional SSC problem*, or *CSSCP*.

Note that the set of influential due dates does not change during the branching process, so it should be obtained only once.

A (*minimum*) *conditional spanning subnetwork* for $\tilde{G}(V_1, \ldots, V_k)$ is a (shortest) spanning subnetwork for $\tilde{G}(V_1, \ldots, V_k)$ that can be extended into a conditional spanning tree for G. (A definition of a spanning subnetwork for $\tilde{G}(V_1, \ldots, V_k)$ was given in Section 7.2.) Observe that a minimum conditional spanning subnetwork for $\tilde{G}(V_1, \ldots, V_k)$ must contain all plus-edges from $\tilde{G}(V_1, \ldots, V_k)$ and no minus-edges, and should not create cycles with plus-edges from $G \backslash \tilde{G}(V_1, \ldots, V_k)$. It can be obtained by the standard greedy algorithm, because cycle-free sets of edges of $\tilde{G}(V_1, \ldots, V_k)$ that do not contain any plus-edges or minus-edges and do not create cycles with plus-edges form a matroid, which can be verified straightforwardly by checking the growth property and the hereditary property (Ahuja et al. 1993, Chap. 13).

The lower bound $B_0''(V_1, \ldots, V_k)$ for CSSCP is defined using the same formula, but we use a minimum conditional spanning subnetwork for $\tilde{G}(V_1, \ldots, V_k)$ instead of a minimum spanning subnetwork. Justification is the same as that for the original $B_0''(V_1, \ldots, V_k)$, using the following auxiliary result.

Lemma 5. *The set of cycle-free subsets of edges of $\tilde{G}(V_1, \ldots, V_k)$ that can be extended to a conditional spanning tree of G forms a matroid.*

Proof. A cycle-free subset of edges of $\tilde{G}(V_1, \ldots, V_k)$ that can be extended to a conditional spanning tree of G will be called an *independent set*, or shortly an *i-set*. To prove that the set of all i-sets forms a matroid, we need to prove the hereditary property and the growth property (Ahuja et al. 1993, Chap. 13). The hereditary property (i.e., that a subset of an i-set is an i-set itself) is obvious. Let us prove the growth property.

Suppose that A and B are i-sets such that $|B| > |A|$. We need to show that there is an $e \in B \backslash A$ such that $A \cup \{e\}$ is an i-set. Since A, B can be extended to a conditional spanning tree of G, they cannot contain minus-edges. Let E^+, A^+, B^+ be the set of all plus-edges and the sets of plus-edges of A and B, respectively. Observe that any i-set remains an i-set after adding some plus-edges from $\tilde{G}(V_1, \ldots, V_k)$.

If $B^+ \setminus A^+$ is nonempty, then adding any edge from $B^+\setminus A^+$ to A produces an i-set, and we are done. So, suppose that $B^+\setminus A^+ = \emptyset$. Then consider $A' = A \cup E^+$ and $B' = B \cup E^+$. Observe that $|B'| > |A'|$, and A' and B' are cycle-free. Since cycle-free subsets of E form a graphic matroid, and since $(B'\setminus A') = (B\setminus A)$ because $B^+\setminus A^+ = \emptyset$, there is an edge $e \in B\setminus A$ such that $A' \cup \{e\}$ is cycle-free. Therefore, $A \cup \{e\}$ is an i-set, and we are done. □

Now we discuss the necessary adjustment of the lower bound $B'_0(V_1, \ldots, V_k)$.

Plus-edges with both endpoints within the same V_i, $i \in [1:k]$, will be called *strict-plus-edges*, and let E^{spe} be the set of all strict-plus-edges and R^{spe} be the sum of the lengths of all strict-plus-edges. Observe that the subnetwork G' from the definition of CSSCP (see also the definition of Problem SSC$(V_1, \ldots, V_k)$) must include all strict-plus-edges, but not necessarily other plus-edges (other plus-edges may be constructed after constructing G'). Consider the network $G^{\text{spe}} = (V, E^{\text{spe}})$ with the set of vertices V and the set of edges E^{spe}. The connected components of this network will be called *strict-plus-components*. Some of the strict-plus-components may be single vertices, and the union of all strict-plus-components is the whole set V. Now, consider the network $G^s = (V^s, E^s)$ obtained from $G = (V, E)$ by two operations:

(a) deleting all minus-edges;

(b) contracting each strict-plus-component into a single vertex.

Edges with endpoints in the same strict-plus-component disappear, but all edges (except for minus-edges) with endpoints in different strict-plus-components remain present. Note that E^s may contain multiedges (several edges joining the same pair of vertices), but all of them are kept present. The subsets $V_1, \ldots, V_k$ of V become the subsets $V_1^s, \ldots, V_k^s$ of V^s. For each $v \in V^s$, define $c'_{\min}(v) = \min\{c_e \mid e \text{ is adjacent to } v \text{ in } G^s\}$, and for each V_i^s define $c'_{\min}(V_i^s) = \min\{c'_{\min}(v) \mid v \in V_i^s\}$, then the adjusted $B'_0(V_1, \ldots, V_k)$ bound is $\sum_{i=1}^k \sum_{v\in V_i^s} c'_{\min}(v) - \sum_{i=1}^k c'_{\min}(V_i^s) + R^{\text{spe}}$. Justification is similar to the justification for the original bound $B'_0(V_1, \ldots, V_k)$.

Adjustment of the lower bound $B_0(V_1, \ldots, V_k)$ is done by combining the approaches for adjustment of $B'_0(V_1, \ldots, V_k)$ and $B''_0(V_1, \ldots, V_k)$. First, we use the minimum conditional spanning subnetwork for $\tilde{G}(V_1, \ldots, V_k)$ instead of the minimum spanning subnetwork to obtain $e_1, e_2, \ldots$. Second, to obtain Ω we do the following:

1. Include the lengths of all strict-plus-edges in Ω (with a slight abuse of terminology, Ω will denote both a sorted list and the set of its elements).

2. Obtain the network $G^s = (V^s, E^s)$ defined above, and the corresponding subsets $V_1^s, \ldots, V_k^s$ of V^s.

3. For all $v \in V^s$, compute $c'_{\min}(v) = \min\{c_e \mid e \text{ is adjacent to } v \text{ in } G^s\}$.

4. For each $i \in [1:k]$, obtain the set $V_i'^s$ by deleting from V_i^s a vertex with the smallest value $c'_{\min}(v)$.

5. Include the values $c'_{\min}(v)$, $v \in \bigcup_{i=1}^k V_i'^s$ into Ω (now Ω has $r(V_1, \ldots, V_k)$ elements).

6. Sort Ω in nonincreasing order (from largest to smallest), obtaining the sorted list $\Omega = (q_1, q_2, \ldots, q_{r(V_1,\ldots,V_k)})$.

Then, the adjusted $B_0^{(t)}(V_1, \ldots, V_k)$ for any $t \in [0: r(V_1, \ldots, V_k)]$ and $B_0(V_1, \ldots, V_k)$ are obtained using the same formulas as before, using the new $e_1, e_2, \ldots$ and $q_1, q_2, \ldots, q_{r(V_1,\ldots,V_k)}$.

The Lagrangean relaxation bound $B_B(V_1, \ldots, V_k)$ is adjusted by finding minimum conditional spanning trees instead of minimum spanning trees in the Lagrangean relaxation subproblems.

8.3. Other Details

Candidate edges are considered for branching in the order of nondecreasing lengths. Each node of the search tree has at most two immediate descendants (sons). If nodes b and c were obtained from a node a by branching on inclusion/exclusion of an edge e to/from the solution, and b corresponds to the inclusion decision while c corresponds to exclusion, then b has priority over c for exploration; also, all descendants of b will have priority over c. In other words, the algorithm uses the depth-first branching strategy with priority for inclusion of an edge over exclusion. The lower bounds are computed (updated) for the nodes obtained by a branching decision to include an edge, and are not computed for the nodes obtained by a branching decision to exclude an edge (any such node inherits the lower bound from its father). If for a node of the search tree the set of plus-edges has size $n-1$, then there can be no further branching from the node since a spanning tree of essential edges has already been obtained, and the exact value of the node is obtained using the algorithm from Section 5. A node of the search tree is fathomed if its lower bound is not smaller than the value of the best discovered solution. The bounds $LB1$, $LB2'$, $LB4$, $LB5(d)$ are adjusted as described in Section 8.2 to be applicable at nodes of the decision tree. The bound $LB3$ was not used in the final version of the algorithm as preliminary experiments indicated that it was typically dominated by other bounds. Also, to compute $B_0(V_1, \ldots, V_k)$ and $LB4$, for simplicity the whole network G was used instead of $\tilde{G}(V_1, \ldots, V_k)$ as described in Remark 4 in Section 7.2.1. Since computing the Lagrangean relaxation bound $B_B(V_1, \ldots, V_k)$ for $LB5(d)$ is time-consuming, at all nodes of the search tree except the root node $LB5(d)$ was computed only for one value of d, namely, the value that produced the best bound $LB5(d)$ at the root node.

9. Computational Experiments

9.1. Computer and Software Details

The algorithms have been coded in ANSI C++ and compiled with the GNU GCC compiler, version 4.4.7. The IBM ILOG CPLEX optimization library version 12.6.0 was used to solve the MILP formulation. The default parameter settings of CPLEX were used, with the exception of the feasibility tolerance limit, which was set to 1E-9 (the default value is 1E-6). This change was introduced to avoid some rounding issues encountered during preliminary computational tests.

The tests were run on a cluster of computers, each having 32 Intel Xeon E5-2670 2.6 GHz. processors and 128 GB RAM running the Linux operating system. The branch-and-bound code does not make use of any form of parallelism, so the 32 processors of each computer were used to solve 32 different instances simultaneously. Memory was not a limiting factor in the experiments with the branch-and-bound algorithm. Therefore, the results can be considered to correspond to a single processor computer with similar clock speed and a smaller amount of RAM.

All instances used for these experiments are publicly available at https://github.com/jordipereiragude/pairwiseConnectionInstances and are described next.

9.2. Random Instances

The first group of instances is generated randomly and will be called *random instances*. For this group, instances of the network G are generated as sparse networks to resemble road networks using the same procedure as in Averbakh and Pereira (2015). For completeness, we quote here the description of this procedure.

> First, n nodes with integer coordinates are selected randomly in the $1{,}000 \times 1{,}000$ Euclidean grid; they will be the nodes of the instance. Then the edges of the instance are generated using the following procedure. In the first stage of the procedure, a spanning tree for the selected nodes is obtained by generating its edges one by one randomly in $n-1$ steps. At each step, the *potential edges* for inclusion in the spanning tree (currently a forest) are the straight-line segments joining pairs of nodes such that they
>
> (a) do not have internal intersections with already included edges;
>
> (b) do not create cycles with already included edges, that is, have endpoints in different connected components.
>
> When the set of potential edges that satisfy conditions (a)–(b) is determined, selection of one of them is done randomly using a probability distribution based on the lengths of the potential edges; a potential edge e has the probability of being selected $K_e / \sum_{e'' \in E''} K_{e''}$, where E'' is the current set of potential edges, and K_e is the reciprocal of the length of e.
>
> In the second stage of the procedure, after a spanning tree has been generated in the first stage, additional $\lceil 0.75n \rceil$ edges are generated one by one. At each step, the set of potential edges is defined by the condition (a) above (condition (b) is no longer applicable). Selection of one potential edge at each step is done randomly using the same probability distribution as above. If at some point there are no potential edges and the instance is not finished yet, the instance is discarded, and the generation process starts from scratch. (Averbakh and Pereira 2015, pp. 721–722)

For a generated instance of network G, the due dates were generated as follows. Let $\hat{k}$ be a fixed integer positive parameter. If $\hat{k} < 0.5(n-1)$, then the set R of relevant pairs is generated by selecting randomly $\hat{k}n$ pairs (i,j), $i,j \in V$, $i < j$ and including them in R. If $\hat{k} \geq 0.5(n-1)$, then all pairs (i,j), $i,j \in V$, $i < j$ are included in R. Thus, $|R|$ grows linearly with n when $n > 2\hat{k}+1$. In our experiments, we used $\hat{k} = 6$. After generating the set R, for each relevant pair the corresponding due date is generated randomly and independently from the uniform distribution over the interval $[MST(1-0.5RDD), MST(1+0.5RDD)]$, where MST is the length of a minimum spanning tree for G, and RDD is a fixed parameter. (The abbreviation RDD stands for the "range of due dates.") We used $RDD \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. For each combination of size n and value RDD, 20 instances were generated.

9.3. Results for Random Instances

Let $d_{\min}$ be the smallest due date. Define

$$gap_L = \frac{UB - LB}{UB + d_{\min}} \cdot 100, \tag{19}$$

where UB and LB are, respectively, the best upper and lower bounds obtained for the instance by the method under consideration (CPLEX or branch and bound). The term $d_{\min}$ is included in the denominator to ensure that gap_L is between 0 and 100. Observe that both LB and UB can be negative for the considered problem, so the value gap_L without $d_{\min}$ in the denominator could be negative or greater than 100. Since $-d_{\min}$ is a trivial lower bound for Z^*, we have $UB \geq -d_{\min}$; also, we can assume that the lower bound LB obtained by the used method is not smaller than $-d_{\min}$, which implies $0 \leq gap_L \leq 100$ for gap_L defined as in (19).

Observe that the denominator of (19) cannot be 0, because we assumed that there is at least one relevant pair, all edge lengths are positive, and $d_{ii} = +\infty$ for all $i \in V$, therefore it is not possible that $UB = -d_{\min}$. Let us discuss additional motivation for using formula (19), in particular the term $d_{\min}$ in the denominator. In Problem NCPC-L, shifting all due dates by the same constant results in an equivalent problem, and bounds UB and LB and the optimal objective value shift (in the opposite direction) by the same constant. Hence, (19) can be interpreted as the standard relative gap for the equivalent problem obtained by adding $(-d_{\min})$ to all due dates, thus making the smallest due date 0.

Table 1. Results for CPLEX Applied to the MILP Formulation for Random Instances with 10-Minute Time Limit

n	5			6			7			8			10		
RDD	No.	$t(s)$	gap_L	No.	$t(s)$	gap_L	No.	$t(s)$	gap_L	No.	$t(s)$	gap_L	No.	$t(s)$	gap_L
0.2	20	0.6	0	20	9.5	0	20	260.7	0	0	600	66.586	0	600	77.058
0.4	20	0.6	0	20	8.1	0	20	173	0	0	600	69.074	0	600	78.868
0.6	20	0.5	0	20	7.2	0	20	230.4	0	0	600	64.86	0	600	77.841
0.8	20	0.5	0	20	6.9	0	17	275.3	7.589	0	600	66.162	0	600	78.629
1	20	0.5	0	20	6.3	0	18	263.7	6.66	0	600	66.351	0	600	77.385

Table 1 provides the results of applying CPLEX to the MILP formulation with a 10 minutes time limit for the random instances. The spanning tree with local search heuristic described in Section 8.1 was used as a warm-start procedure. For each combination of the size and the *RDD* value, we report the number of instances solved to proven optimality (out of 20; column "No."), the average processor time used in seconds (column "$t(s)$"), and the average value of gap_L. The results indicate that the MILP formulation can be used only for very small instances. Increasing the time limit did not result in a significant improvement of performance, and we omit these results here.

Tables 2 and 3 provide results for the branch-and-bound algorithm applied to the random instances with time limits 10 minutes and 10 hours, respectively. In addition to the values reported in Table 1, we also provide the average numbers of explored nodes of the decision tree (in millions; column "Nodes"). The results indicate that the developed branch-and-bound algorithm is reasonably effective for medium-size instances (up to 40–50 vertices). Also, the results show that the algorithm's performance depends strongly on the degree of variability of the due dates (parameter *RDD*); typically, the larger variability, the more difficult is the instance for the algorithm. For instances with low

Table 2. Results for the Branch-and-Bound Algorithm for Random Instances with 10-Minute Time Limit

RDD	No.	$t(s)$	Nodes	gap_L	No.	$t(s)$	Nodes	gap_L	No.	$t(s)$	Nodes	gap_L
n			12				15				20	
0.2	20	0	0	0	20	0	0	0	20	0	0	0
0.4	20	0	0	0	20	0	0	0	20	0	0.1	0
0.6	20	0	0	0	20	0	0.1	0	20	0	0.4	0
0.8	20	0	0.1	0	20	0	0.1	0	20	0.1	1	0
1	20	0	0	0	20	0	0.2	0	20	0.2	2.1	0
n			25				30				35	
0.2	20	0	0.2	0	20	0.1	0.4	0	20	0.1	0.4	0
0.4	20	0.1	1	0	20	0.2	1	0	20	0.1	0.5	0
0.6	20	8.3	144.4	0	20	4.6	21	0	20	16.5	58.5	0
0.8	20	2.8	30.5	0	20	6.1	22	0	20	40.7	307.9	0
1	20	2.7	23.3	0	19	72.6	488.7	0.777	17	162	590.9	1.306
n			40				45				50	
0.2	20	0.1	0.5	0	20	0.5	1.5	0	19	30.3	61.5	0.087
0.4	19	46.7	286.2	0.217	19	33.4	90.5	0.149	18	95.4	184.9	0.351
0.6	19	40	78.7	0.173	18	95.6	121.9	0.796	14	196.1	222.3	1.525
0.8	16	149	267.9	1.429	12	292.6	503.9	2.958	9	409.2	662.5	3.277
1	15	210	418.6	2.14	11	293.1	374.4	3.302	4	513.7	499.7	5.701
n			55				60				70	
0.2	20	1.9	5.5	0	20	20.7	55.2	0	16	146.8	195.7	0.277
0.4	14	194.1	348.9	1.283	14	236.6	294.1	0.662	9	339.4	233.1	1.451
0.6	10	351.9	665.3	2.456	9	379.4	391.3	2.952	6	422.7	337.3	2.674
0.8	4	502.4	531.3	4.163	2	567.7	607.4	4.166	4	484	277.5	4.163
1	2	553.6	500.6	7.194	5	481.5	392.3	4.937	3	511.2	258.2	4.844
n			80				90				100	
0.2	17	198.6	136.6	0.213	13	225.8	176.2	0.433	12	299.9	141.2	0.303
0.4	6	450.6	289.8	1.394	6	428.4	189	1.557	9	407.2	156.3	0.969
0.6	1	570	316.7	3.556	3	523.6	192.6	3.205	0	600	156.6	3.744
0.8	4	488.9	171.1	3.765	1	571.5	168.9	4.462	2	567.8	107.6	5.099
1	0	600	198.1	6.129	0	600	113.5	6.441	0	600	114.9	7.198

Table 3. Results for the Branch-and-Bound Algorithm for Random Instances with 10-Hour Time Limit

RDD	No.	$t(s)$	Nodes	gap_L	No.	$t(s)$	Nodes	gap_L	No.	$t(s)$	Nodes	gap_L
n		25				30				35		
0.2	20	0	0.2	0	20	0.1	0.4	0	20	0.1	0.4	0
0.4	20	0.1	1	0	20	0.2	1	0	20	0.1	0.5	0
0.6	20	8.3	144.4	0	20	4.6	21	0	20	16.5	58.5	0
0.8	20	2.8	30.5	0	20	6.1	22	0	20	40.7	307.9	0
1	20	2.7	23.3	0	20	91.2	749.1	0	20	1,099	6,118.4	0
n		40				45				50		
0.2	20	0.1	0.5	0	20	0.5	1.5	0	20	52.9	108.1	0
0.4	20	93.4	686.6	0	20	38.9	110.4	0	20	292.2	556.7	0
0.6	20	43.9	84.5	0	19	2,163.7	4,209	0.474	16	7,523.1	9,517.6	1.264
0.8	19	2,703.4	3,964.3	0.445	18	4,681.9	11,532.8	1.19	15	11,998.9	22,365	2.048
1	18	5,554.8	11,883.8	0.788	17	9,506.8	11,864.3	1.35	11	18,629.3	18,208.3	3.981
n		55				60				70		
0.2	20	1.9	5.5	0	20	20.7	55.2	0	20	1,030.1	740.7	0
0.4	18	5,870.3	10,901.5	0.437	19	3,161.5	3,034	0.213	14	11,541.8	9,649	1.062
0.6	15	9,968.2	14,775.1	1.507	12	14,642.2	20,812.3	2.571	10	19,891.6	17,579.8	2.287
0.8	13	17,611	18,084.6	2.461	9	25,962	22,299.2	2.763	5	27,490.9	14,252	3.986
1	4	29,452.8	30,440.3	6.695	9	21,516.6	19,453.5	4.413	5	29,480.7	13,899.6	4.573
n		80				90				100		
0.2	19	2,346.6	1,865.5	0.123	19	4,327.4	2,684.3	0.117	16	7,950.8	4,229	0.215
0.4	14	16,006.8	11,243.6	0.789	9	22,018.8	11,420	1.361	12	16,954.5	5,426.9	0.817
0.6	5	29,388.4	18,880	3.225	5	28,319	11,859.1	3.059	0	36,000	10,941.7	3.717
0.8	4	28,808.6	10,091.2	3.729	1	34,201.3	10,203.5	4.454	2	32,427.2	6,757.5	5.099
1	2	32,691.6	10,806.4	6.01	0	36,000	6,746.7	6.333	0	36,000	8,184.3	7.168

variability of due dates, the algorithm is effective also for larger sizes.

Define values

$$gap_L^{UB_0} = \frac{UB_0 - Z^*}{UB_0 + d_{\min}} \cdot 100,$$

$$gap_L^{UB-MST} = \frac{UB_{MST} - Z^*}{UB_{MST} + d_{\min}} \cdot 100,$$

where UB_0 is the initial upper bound, that is, the objective value of the solution obtained by the spanning tree with local search heuristic, and UB_{MST} is the objective value of the solution obtained by the minimum spanning tree heuristic (both heuristics are described in Section 8.1). Table 4 reports these values averaged over all random instances of the corresponding size that are solved to proven optimality. We observe that the heuristics produce quite good solutions, and that the local search is quite effective in improving the quality of the solutions obtained by the minimum spanning tree heuristic. Although values $gap_L^{UB_0}$ and gap_L^{UB-MST} appear to decrease when the instance size n grows, this is likely to be due to the decreasing proportion of instances solved to proven optimality; remember that $gap_L^{UB_0}$ and gap_L^{UB-MST} are averaged only over such "easier" instances, so no conclusions about the dynamics of $gap_L^{UB_0}$ and gap_L^{UB-MST} as n grows can be made. We note that the heuristics are very fast and typically take only a fraction of a second.

We also conducted additional experiments to understand the relative strength and contribution of individual lower bounds. These results are presented and discussed in the online supplement to the paper. The main conclusion is that the contributions of individual bounds are inseparable, and the algorithm in the paper that uses all of them (except $LB3$) performs much better than any version that uses only one bound. Another set of additional experiments was conducted to understand the effect of network density on the performance of the branch-and-bound algorithm, by varying the

Table 4. Heuristic Results

n	No.	$gap_L^{UB_0}$	gap_L^{UB-MST}
12	100	0.17	1.24
15	100	0.18	1.29
20	100	0.05	1.32
25	100	0.08	1.06
30	100	0.14	0.98
35	100	0.09	1
40	97	0.08	1.1
45	94	0.04	1.07
50	82	0.1	1.06
55	70	0.08	0.88
60	69	0.06	0.75
70	54	0.03	0.63
80	44	0.05	0.83
90	34	0.02	0.46
100	30	0.03	0.34

number of edges generated at the second stage of the instance-generating procedure described in Section 9.2. In general, as expected, increasing (decreasing) the network density makes the problem somewhat harder (easier), apparently because of the increase (decrease) of the branching factor and the corresponding increase (decrease) of the size of the decision tree. Note that real-life transportation networks are almost always planar and therefore sparse (a well-known corollary from Euler's formula for planar graphs is that in a planar graph with n vertices, the number of edges is not greater than $3n-6$ (West 1996, Chap. 7). We believe that for typical real-life transportation networks, the number of edges ranges approximately from n (the case of cycle-free networks) to $2n$ (the case of a grid). For the random instances used in the experiments reported in the paper, the number of edges is about $1.75n$.

9.4. Chilean Instances

The second group of instances was based on data for 2010 Chile earthquake (February 27, 2010; magnitude 8.8 on the moment magnitude scale; affected area contained around 80% of the total population of Chile). The earthquake seriously damaged the transportation infrastructure; the related emergency and reconstruction operations cost the Chilean government over 150 billion Chilean pesos. We obtained detailed information about the damages, reconstruction works conducted by the government, and their budgets, from the Chilean government and CIGIDEN, the Chilean Center for Natural Disaster Management, (Chilean Ministry of Public Works 2010, 2011). Based on this information, we obtained a *reconstruction network* for our problem. The same reconstruction network was used in Averbakh and Pereira (2015) for generating some instances; for completeness, we quote here the description from Averbakh and Pereira (2015) of the procedure of obtaining the reconstruction network.

> First, we identified all continental Chilean cities. According to the Chilean government, any municipality with over 5,000 inhabitants is considered to be a city. Two hundred and twenty eight cities were identified, along with their population sizes.
>
> Second, we identified the network of major public roads between the cities. We call this network a *base network*.
>
> Third, we identified the roads that required reconstruction work. These roads will be called *affected*.
>
> Fourth, we identified the connected components of the network obtained from the base network by deleting the affected roads. These connected components are the vertices of the reconstruction network . . . . The affected edges of the base network that connect different components are the edges of the reconstruction network; as the 'length' of such an edge, we take the reconstruction expenditure for the corresponding road. In our opinion, the reconstruction expenditure for a road is a better proxy for the time required to repair the road than the actual length of the road. If several affected edges connect two components, we take one with the smallest reconstruction expenditure . . . . The reconstruction network has 53 vertices and 76 edges.
>
> (Averbakh and Pereira 2015, p. 728)

Using the obtained reconstruction network, we generated 100 instances, which we call *Chilean instances*. The reconstruction network is used as network G for all Chilean instances, which differ from each other only by the due dates. The process of generating the relevant pairs and the due dates is the same as for the random instances; for each value of parameter *RDD* from $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, 20 instances are generated.

Table 5 provides the results of applying the developed branch-and-bound algorithm to the Chilean instances, with time limits 10 minutes and 10 hours. Again, we see that instances with higher degree of variability in due dates are typically more difficult for the algorithm. However, overall the algorithm was quite successful in solving Chilean instances. On the contrary, CPLEX applied to the MILP formulation could not solve any Chilean instances (as expected).

We also applied the branch-and-bound algorithm without imposing a time limit to the five Chilean instances not solved to proven optimality within the 10 hours limit. All of them were successfully solved to proven optimality, with the following solution times: 72 hours for the instance with RDD = 0.8, and 12, 67, 145, and 162 hours for the four instances with RDD = 1.0, respectively.

Table 5 also reports the values $gap_L^{UB_0}$ and gap_L^{UB-MST} averaged over all 20 instances for the corresponding

Table 5. Branch-and-Bound and Heuristic Results for Chilean Instances

Time limit	600 s				36,000 s					
RDD	No.	$t(s)$	Nodes	gap_L	No.	$t(s)$	Nodes	gap_L	$gap_L^{UB_0}$	gap_L^{UB-MST}
0.2	20	0.4	1.4	0	20	0.4	1.4	0	0	0.02
0.4	18	77.3	276.7	0.319	20	148.5	447	0	0	0.07
0.6	15	187.8	256.9	0.884	20	1,736.6	1,797.9	0	0	0.07
0.8	16	166.2	380.6	1.197	19	2,908.3	5,682.4	0.479	0	0.06
1	11	315.1	501.4	2.424	16	8,307.8	9,399.6	1.535	0	0.1

RDD. We observe that the performance of the heuristics is excellent for Chilean instances. In fact, for all Chilean instances in this experiment, the spanning tree with local search heuristic found optimal solutions, and branching was required only to confirm optimality. Apparently, the structure of the reconstruction network (e.g., presence of some "junction" nodes whose deletion would decompose the network into disconnected parts, sparsity, etc.) makes it more effectively approximated by spanning trees than the networks in the random instances. We note that in additional experiments we encountered some instances based on the reconstruction network where the spanning tree with local search heuristic did not obtain an optimal solution, but this happened quite rarely.

10. Conclusion

We introduced and studied a new lateness-based network construction problem. The problem is more general than the lateness-based network construction problem from Averbakh and Pereira (2015), because (a) connections between any pairs of vertices may be important, not only the connections to a specified vertex (depot) as in Averbakh and Pereira (2015); (b) all areas of the network are accessible for construction activities at any time, so the set of constructed edges does not have to be always connected as in Averbakh and Pereira (2015). We presented a polynomial algorithm for cycle-free networks, an MILP formulation for general networks, a number of lower bounds exploiting the combinatorial structure of the problem, and a branch-and-bound algorithm. The structure of the branch-and-bound algorithm is entirely different from that of Averbakh and Pereira (2015), as the branching is done on network design decisions (inclusion/exclusion of an edge in the set of essential edges) but not on sequencing decisions. Also, the lower bounds in this paper exploit different ideas than those of Averbakh and Pereira (2015), for example, the Lagrangean relaxation-based bound $LB5(d)$, or the bound $LB4$ that uses matroidal properties. At the same time, the bounds $LB1$ and $LB3$ are similar to the bounds used in Averbakh and Pereira (2015).

The computational experiments indicate that while the MILP formulation with CPLEX can be used only for very small instances, the branch-and-bound algorithm can be used for medium-size instances. The performance of the branch-and-bound algorithm depends on the degree of variability of the due dates; typically, the larger the variability, the more difficult is the instance for the algorithm. The branch-and-boundalgorithm was quite successful in solving instances based on the real-life 2010 Chilean earthquake data.

A possible direction for future research is to study pairwise connectivity restoration problems with different types of scheduling objectives.

Acknowledgments

The authors thank the CIGIDEN (Chilean Center for Natural Disaster Management) and the Chilean Ministry of Public Works for providing data on restoration works for 2010 Chilean earthquake.

References

Ahuja R, Magnanti T, Orlin J (1993) *Network Flows* (Prentice-Hall, Upper Saddle River, NJ).

Averbakh I (2012) Emergency path restoration problems. *Discrete Optim.* 9(1):58–64.

Averbakh I (2017) Minimizing the makespan in multiserver network restoration problems. *Networks* 70(1):60–68.

Averbakh I, Pereira J (2012) The flowtime network construction problem. *IIE Trans.* 44(8):681–694.

Averbakh I, Pereira J (2015) Network construction problems with due dates. *Eur. J. Oper. Res.* 244(3):715–729.

Baxter M, Elgindy T, Ernst A, Kalinowski T, Savelsbergh M (2014) Incremental network design with shortest paths. *Eur. J. Oper. Res.* 238(3):675–684.

Beasley J (1989) An SST-based algorithm for the Steiner problem in graphs. *Networks* 19(1):1–16.

Chilean Ministry of Public Works (2010) Informe de seguimiento del programa de emergencia y reconstrucción de infraestructura del ministerio de obras públicas, December 2010. Technical report, Government of Chile, Santiago.

Chilean Ministry of Public Works (2011) Informe de reconstrucción, MOP, 4 de febrero del 2011. Technical report, Government of Chile, Santiago.

Engel K, Kalinowski T, Savelsbergh M (2017) Incremental network design with minimum spanning trees. *J. Graph Algorithms Appl.* 21(4):417–432.

Garey M, Johnson D (1979) *Computers and Intractability* (Freeman, New York).

Guha S, Moss A, Naor J, Schieber B (1999) Efficient recovery from power outage (extended abstract). Vitter J, Larmore L, Leighton T, eds. *Proc. Thirty-First Annual ACM Sympos. Theory Comput.* STOC '99 (Association for Computing Machinery, New York), 574–582.

Kalinowski T, Matsypura D, Savelsbergh M (2015) Incremental network design with maximum flows. *Eur. J. Oper. Res.* 242(1):51–62.

Nurre S, Sharkey T (2014) Integrated network design and scheduling problems with parallel identical machines: Complexity results and dispatching rules. *Networks* 63(4):306–326.

Nurre S, Cavdaroglu B, Mitchell J, Sharkey T, Wallace W (2012) Restoring infrastructure systems: An integrated network design and scheduling (INDS) problem. *Eur. J. Oper. Res.* 223(3): 794–806.

Shore M, Foulds L, Gibbons P (1982) An algorithm for the Steiner problem in graphs. *Networks* 12(3):323–333.

Sousa J, Wolsey L (1992) A time indexed formulation of non-preemptive single machine scheduling problems. *Math. Programming* 54(1):353–367.

West D (1996) *Introduction to Graph Theory* (Prentice-Hall, Upper Saddle River, NJ).