Technical Report AC-TR-18-001

first version: Feb 2018

Exact approaches for the directed network design problem with relays

Markus Leitner, Ivana Ljubić, Martin Riedler, and Mario Ruthmair
Exact approaches for the directed network design problem with relays

Markus Leitner\textsuperscript{a}, Ivana Ljubi\textsuperscript{c}, Martin Riedler\textsuperscript{c,*}, Mario Ruthmair\textsuperscript{a}

\textsuperscript{a}University of Vienna, Department of Statistics and Operations Research, Vienna, Austria
\textsuperscript{b}ESSEC Business School of Paris, Cergy-Pontoise, France
\textsuperscript{c}Institute of Logic and Computation, TU Wien, Vienna, Austria

Abstract

We study the directed network design problem with relays (DNDPR) whose aim is to construct a minimum cost network that enables the communication of a given set of origin-destination pairs. Thereby, expensive signal regeneration devices need to be placed to cover communication distances exceeding a predefined threshold. Applications of the DNDPR arise in telecommunications and transportation. We propose two new integer programming formulations for the DNDPR. The first one is a flow-based formulation with a pseudo-polynomial number of variables and constraints and the second is a cut-based formulation with an exponential number of constraints. For the latter we develop a branch-and-cut algorithm and also consider valid inequalities to strengthen the obtained dual bounds and to speed up convergence. The results of our extensive computational study on diverse sets of benchmark instances show that our algorithms outperform the previous state-of-the-art method based on column generation.

Keywords: Integer Programming, Networks, Layered Graphs, Telecommunications

1. Introduction

The directed network design problem with relays (DNDPR) was introduced by Li et al. [14] for modeling the design of networks when the maximum distance a commodity (i.e., signal) can travel is bounded from above by some threshold. This distance limit can be surpassed by locating special, commodity regenerating equipment (relays) at intermediate network nodes. Applications of this problem arise in the design of transportation and telecommunication networks [14]. In the latter, signals deteriorate after traveling a certain distance and thus there is the need to regenerate them before a predefined maximum distance is exceeded. Thus, comparably expensive regenerating devices (e.g., repeaters) need to be installed, see, e.g., Cabral et al. [1], Chen et al. [2], in order to avoid signal loss or falsification of the transmitted information.

The DNDPR is defined on a digraph $G = (V,A,c,w,d)$ with relay costs $c: V \to \mathbb{Z}_{\geq 0}$, arc costs $w: A \to \mathbb{Z}_{\geq 0}$, and arc distances $d: A \to \mathbb{Z}_{\geq 0}$. Moreover,
a distance bound \( \lambda_{\text{max}} \) and a set of commodity pairs \( K \) are given. For each commodity \((u, v) \in K\), nodes \( u \) and \( v \) are called its source and target, respectively. The goal of the DNDPR is to place relays on a subset of the nodes \( V' \subseteq V \) and to select a subset of arcs \( A' \subseteq A \) such that:

1. The subgraph induced by \( A' \) contains for each \((u, v) \in K\) a directed (simple) path from \( u \) to \( v \) not exceeding the distance limit between \( u \) and the first relay, any two consecutive relays, and the last relay and \( v \), and
2. the cost induced by installing relays and arcs, defined as

\[
\sum_{v \in V'} c_v + \sum_{a \in A'} w_a
\]

is minimized.

A problem instance and its optimal solution are given in Figure 1.

![Diagram](image.png)

Figure 1: Example instance with two commodities \( K = \{ (0, 3), (0, 4) \} \) and \( \lambda_{\text{max}} = 7 \). Arc distances are provided next to the arcs, relay and arc costs are given in parentheses. Relays and arcs used in the optimal solution are marked bold and blue.

To simplify notation, we will in the following use \( S = \{ u \mid (u, v) \in K \} \) to denote the set of commodity sources and \( T^n = \{ v \mid (u, v) \in K \} \) to denote all targets that need to be reached from source \( u \in S \). Additionally, \( \delta^{-}(W) = \{ (j, i) \mid i \in W, (j, i) \in A \} \) and \( \delta^{+}(W) = \{ (i, j) \mid i \in W, (i, j) \in A \} \) will be used to denote the sets of incoming and outgoing arcs for node sets \( W \subseteq V \). By slightly abusing notation, we write \( \delta^{-}(i) \) and \( \delta^{+}(i) \) instead of \( \delta^{-}(\{i\}) \) or \( \delta^{+}(\{i\}) \) for single node sets \( W = \{i\} \).

Related work. The DNDPR has been introduced in Li et al. [14] where a compact node-arc formulation and an arc-path formulation with an exponential number of variables have been proposed. Two branch-and-price (B&P) algorithms based on the latter formulation have been developed that differ in the way the pricing subproblem is solved. A metaheuristic based on tabu search has been recently proposed in Li et al. [15].
Several related studies consider the undirected variant of the problem, i.e., the network design problem with relays (NDPR). The NDPR has been introduced in Cabral et al. [1] where the proposed B&P approach turned out to be quite inefficient (even for small instances) due to the high complexity of the associated pricing subproblem. Therefore, Cabral et al. [1] have focused on construction heuristics that were able to tackle larger problem instances in comparably short time. More efficient B&P approaches for the NDPR have been given in Leitner et al. [13] and Yıldız et al. [22]. In addition to these exact approaches, several metaheuristics have been developed for approximately solving larger problem instances: genetic algorithms (Kulturel-Konak and Konak [11], Konak [10]), tabu search (Lin et al. [16]), and variable neighborhood search (Xiao and Konak [21]).

Existing methods for the NDPR cannot be applied in a straightforward way to the DNDPR, since NDPR solutions may contain cycles (or even traverse a single edge in both directions for one commodity). Besides, asymmetric arc costs and arcs existing in a single direction only would require some adaptations.

**Node-arc formulation.** The node-arc formulation (1) introduced in Li et al. [14] is used for comparison purposes in our computational study. Therefore, we briefly summarize it in the following. Its basic idea is to keep track of the distance from the last relay (or the source of the respective commodity) in order to forbid subpaths exceeding the distance bound. Four sets of variables are used. Binary arc and node variables \( x_{a}, \forall a \in A \), and \( y_{i}, \forall i \in V \), mark the selected arcs and relays, respectively. For each commodity \( k = (u,v) \in K \) and node \( i \in V \), continuous variable \( v_{k}^{i} \) tracks the distance of node \( i \) to the preceding relay or the source \( u \) of that commodity (in case the path from \( u \) to \( i \) does not contain relays). Finally, multi-commodity flow variables \( f_{a}^{k}, \forall k \in K, \forall a \in A \), are used to enforce connectivity of each commodity pair.

\[
\begin{align*}
\min & \sum_{i \in V} c_{i}y_{i} + \sum_{a \in A} w_{a}x_{a} \\
\text{s.t.} & \sum_{a \in \delta^{+}(i)} f_{a}^{k} - \sum_{a \in \delta^{-}(i)} f_{a}^{k} = \begin{cases} 1 & \text{if } k = (i,j) \\ -1 & \text{if } k = (j,i) \\ 0 & \text{otherwise} \end{cases} \forall k \in K, \forall i \in V \\
& f_{a}^{k} \leq x_{ij} \forall k \in K, \forall (i,j) \in A \\
& v_{i}^{k} + d_{ij}f_{ij}^{k} - \lambda_{\max}(1 - f_{ij}^{k} + y_{j}) \leq v_{j}^{k} \forall k \in K, \forall (i,j) \in A \\
& v_{i}^{k} + d_{ij}f_{ij}^{k} \leq \lambda_{\max} \forall k \in K, \forall (i,j) \in A \\
& 0 \leq v_{i}^{k} \leq \lambda_{\max}(1 - y_{i}) \forall k \in K, \forall i \in V \\
& v_{u,v}^{k} = 0 \forall (u,v) \in K \\
& f_{a}^{k} \in \{0,1\} \forall k \in K, \forall (i,j) \in A \\
& y_{i} \in \{0,1\} \forall i \in V \\
& 0 \leq x_{ij} \leq 1 \forall (i,j) \in A 
\end{align*}
\]

Flow conservation constraints (1b) together with linking constraints (1c) ensure the existence of a directed path from \( u \) to \( v \) for each commodity pair \((u,v) \in K\). Constraints (1d) ensure that the value of variable \( v_{k}^{i} \) is at least the
distance from the last relay or from the source, respectively, along the path connecting commodity \( k \in K \). The distance limit is enforced using inequalities (1e) and (1f). The latter inequalities also link distance and relay variables. Observe that binary (rather than continuous) flow variables are needed to prevent flow splittings which would yield incorrect values of distance variables \( v \). The main weakness of this model is its relatively weak linear programming (LP) relaxation bound, resulting from the (potentially) large coefficient \( \lambda_{\text{max}} \) required in constraints (1d).

**Overview and contributions.** Two mixed integer linear programming (MILP) formulations that are based on considering one layered graph per source are introduced in Section 2. The first one is a flow-based formulation with a pseudo-polynomial number of variables and constraints, whereas the second one uses an exponential number of connectivity constraints. Subsequently, different families of symmetry breaking constraints and valid inequalities are introduced. Section 3 describes components and variants of a branch-and-cut algorithm based on the second formulation, introduces preprocessing routines, and details a heuristic used to obtain initial solutions. Benchmark instances used in our study are described in Section 4 where we also verify the effectiveness of our algorithms by extensive computational experiments.

2. Formulations

**Properties of Feasible Solutions.** In the DNDPR, the routing of each single commodity is done following a simple path, see [14]. Hence, the in-degree of each node in the routing path is at most one. An optimal solution is a union of all routing paths over all commodity pairs, and hence, the in-degree of a node in this solution can be as large as the number of commodities. On the other hand, if all commodities share a common source node, then it is not difficult to see that there always exists an optimal solution in which the in-degree of each node is at most one, i.e., such that the set \( A' \) of selected arcs forms an arborescence.

**Theorem 1.** If \( S = \{ u \} \), there exists an optimal DNDPR solution which corresponds to a Steiner arborescence rooted at \( u \), whose leaves are a subset of the nodes from \( T^u \).

Theorem 1 enables the interpretation of an optimal solution as a union of several Steiner arborescences (one per source). It does not help, however, to handle the distance constraints and the installation of relays to respect the threshold \( \lambda_{\text{max}} \). To deal with these issues, we propose to exploit layered graphs introduced below.

The new formulations presented in this article use extended, so called layered graphs. The basic idea of layered graphs is to introduce multiple copies for each node and arc of an original graph along one or multiple dimensions (e.g., time or distance) to implicitly model certain constraints and to obtain stronger mathematical models. In our case, layered graphs are used to encode the distances, so that only feasible paths with respect to the distance bound \( \lambda_{\text{max}} \) are generated. Picard and Queyranne [19] were among the first to consider layered graphs for solving the time-dependent traveling salesman problem. More recent successful applications of layered graphs are, e.g., given in Godinho et al.
For the DNDPR, we construct layered digraph $G_L = (V_L, A_L)$ whose node set $V_L$ is recursively defined by sets $V^l_L$, $l \in \{0, 1, \ldots, \lambda_{\text{max}}\}$. Thereby, $V_L = V^\lambda_{\text{max}}_L$ and each subset $V^l_L$ contains all nodes that can be reached with a total distance of at most $l$ starting from a node at layer zero, i.e.,

$$V^0_L = \{i_0 \mid i \in V\}$$
$$V^l_L = \{i_m \in V^{l-1}_L, (i, j) \in \delta^+(i), m + d(i, j) = l\} \cup V^{l-1}_L$$

Arc set $A_L$ contains one arc $(i_l, j_m)$ for each pair of nodes $i_l, j_m \in V_L$ that is connected in the original graph $G$, i.e., for which $(i, j) \in A$. Furthermore, arcs $(i_l, i_0)$ are included for each node $i_l \in V_L$ that is not at layer zero, i.e., when $l > 0$. As the latter arcs correspond to using a relay, we will call them relay arcs. Formally, arc set $A_L = A^a_L \cup A^r_L$, where $A^r_L$ is the set of relay arcs and $A^a_L$ is the set of arcs derived from the original graph:

$$A^r_L = \{(i_l, i_0) \mid i_l \in V_L, l > 0\}$$
$$A^a_L = \{(i_l, j_m) \mid i_l, j_m \in V_L, (i, j) \in A, d(i, j) = m - l\}$$

Figure 2 shows the layered graph corresponding to the instance given in Figure 1 as well as the embedding of the optimal solution in the layered graph. Thereby, relay arcs $A^r_L$ are depicted in dashed lines and the remaining arcs in solid lines. Bold blue arcs indicate those that are included in the considered solution.

![Layered graph](image)

Figure 2: Layered graph $G_L = (V_L, A_L)$ for $\lambda_{\text{max}} = 7$ corresponding to the instance in Figure 1. The optimal solution is marked bold and blue.

### 2.1. Multi-commodity flow formulation

The layered multi-commodity flow formulation ($\text{LMCF}$) is based on flow variables $f^{uv}_a \geq 0$, $\forall (u, v) \in K$, $\forall a \in A_L$. As in the node-arc formulation, variables $y_i \in \{0, 1\}, \forall i \in V$, indicate whether a relay is placed at some node and variables $x_a, \forall a \in A$, indicate whether an arc is included in a solution.
min \sum_{i \in V} c_i y_i + \sum_{a \in A} w_a x_a \quad (2a)

s.t. \sum_{a \in \delta^+(u_0)} f_{uv}^a = 1 \quad \forall (u, v) \in \mathcal{K} \quad (2b)
\sum_{a \in \delta^-(i_l)} f_{uv}^a - \sum_{a \in \delta^+(i_l)} f_{uv}^a = 0 \quad \forall (u, v) \in \mathcal{K}, \forall i_l \in V_L : i \notin \{u, v\} \quad (2c)
\sum_{a \in \delta^-(i_l)} f_{uv}^a = 1 \quad \forall (u, v) \in \mathcal{K} \quad (2d)
\sum_{a \in \delta^-(i_l)} f_{uv}^a \leq 1 \quad \forall (u, v) \in \mathcal{K}, \forall i \in V \setminus \{u, v\} \quad (2e)
\sum_{a \in (i_l, j_m) \in A^L_L} f_{uv}^a \leq y_i \quad \forall (u, v) \in \mathcal{K}, \forall i, j \in A \quad (2f)
\sum_{a \in (i_l, j_m) \in A^L_L} f_{uv}^a \leq x_{ij} \quad \forall (u, v) \in \mathcal{K}, \forall i, j \in A \quad (2g)
\forall i \in V \quad (2h)
\forall a \in A \quad (2i)
0 \leq f_a^u \leq 1 \quad \forall (u, v) \in \mathcal{K}, \forall a \in A_L \quad (2j)

For each commodity, flow balance constraints (2b)-(2d) together with linking constraints (2g) ensure connectivity between the source and exactly one copy of the target node. Inequalities (2e) ensure that this connection contains at most one copy of each intermediate node, i.e., that the associated path in the original graph is simple. Finally, constraints (2f) link the relay arcs to the relay variables.

2.2. Cut formulation

In contrast to formulation (L_MCF) which considers one variable for each commodity pair and layered graph arc, the layered cut formulation (L_CUT) uses one layered graph variable \( z_a^u \in \{0, 1\} \) for each source \( u \in S \) and layered graph arc \( a \in A_L \). By means of an exponential number of connectivity constraints, each set of variables \( z^u \) will model an arborescence rooted at \( u \) in \( S \) that contains all targets \( v \in T^u \), cf. Gouveia et al. [8] where a similar idea has been used in the context of a Steiner tree problem with multiple root nodes.

The L_CUT formulation reads as follows:

\[
\min \sum_{i \in V} c_i y_i + \sum_{a \in A} w_a x_a \quad (3a)
\]

s.t. \sum_{a \in \delta^-(W)} z_a^u \geq 1 \quad \forall u \in S, \forall W \subseteq V_L \setminus \{u_0\}, \exists v \in T^u : \{v_l \in V_L\} \subseteq W \quad (3b)
\sum_{i_l \in V_L, a \in \delta^-(i_l) \setminus A^L_L} z_a^u = 1 \quad \forall u \in S, \forall i \in T^u \quad (3c)
\sum_{i_l \in V_L, a \in \delta^-(i_l) \setminus A^L_L} z_a^u \leq 1 \quad \forall u \in S, \forall i \in V \setminus (T^u \cup \{u\}) \quad (3d)
\[ \sum_{a=(i_l,i_0) \in A_L^u} z_u^a \leq y_i \quad \forall u \in S, \forall i \in V \quad (3e) \]
\[ \sum_{a=(i_l,j_m) \in A_a} z_u^a \leq x_{ij} \quad \forall u \in S, \forall (i,j) \in A \quad (3f) \]
\[ z_u^a \in \{0,1\} \quad \forall u \in S, \forall a \in A_L \quad (3g) \]
\[ y_i \in \{0,1\} \quad \forall i \in V \quad (3h) \]
\[ x_a \in \{0,1\} \quad \forall a \in A \quad (3i) \]

Connectivity constraints (3b) state that every subset of nodes containing all copies of some target node must be connected to the corresponding source. As there are exponentially many of these constraints, we will add them on the fly in a cutting plane approach, see Section 3.3 for details. Constraints (3c) and (3d) prevent cycles by ensuring that each target node is visited exactly once and each non-target node is visited at most once. Thus, together with constraints (3b) they ensure that, for every source \( u \in S \), the subgraph induced by all arcs \( a \in A_L \) such that \( z_u^a = 1 \) is an arborescence rooted at \( u_0 \) that contains exactly one copy of each node \( v \in T^u \). The layered graph variables are linked to the relay node and arc variables on the original graph by inequalities (3e) and (3f).

**Theorem 2.** Formulations (LMCF) and (LCUT) are equally strong, i.e., the LP-relaxation values of the two models coincide.

**Proof.** Let \((x^*, y^*, f^*)\) be an optimal LP-solution of the (LMCF) model. We show how to construct a feasible solution \((\tilde{x}, \tilde{y}, \tilde{z})\) of the (LCUT) model with the same objective value. We set \( \tilde{x} = x^* \), \( \tilde{y} = y^* \), and
\[ \tilde{z}_u^a = \max_{(u,v) \in K} f_{uv}^a \quad \forall u \in S, \forall a \in A_L. \]
Following this definition, it is not difficult to see that flow-based capacity constraints (2f) and (2g) imply constraints (3e) and (3f), respectively. Consider a node \( u \in S \). The flow-balance constraints (2d) are slightly different from the classical ones, due to the aggregation of the incoming flow at the target node \( v \in T^u \). In this constraint the incoming flow is aggregated over all copies \( v_l \in V_L \) of the target node \( v \in T^u \). This can be interpreted as a flow-balance constraint in a modified layered graph, say \( G_L^w \), in which a target node \( t_v \) is introduced for each node \( v \in T^u \), and arcs \((v_l, t_v)\) with infinite capacity are added to this graph. Hence, in such a modified graph, the flow-balance constraints (2b)–(2d) guarantee existence of a path from \( u_0 \) to \( t_v \), for each \( t \in T^u \). By the max-flow min-cut theorem, this implies that cut-set inequalities (3b) are satisfied. Degree-constraints (3c) and (3d) are not satisfied by an arbitrary flow \( f^* \), but the flow can be rerouted (without changing the capacities given by \( x^* \) and \( y^* \)) so that these constraints are always satisfied.

Consider now an optimal LP-solution \((\tilde{x}, \tilde{y}, \tilde{z})\) of the (LCUT) model. For each commodity pair \((u,v) \in K\), we consider the graph \( G_L^{uw} \) described above, with arc capacities \( c_{ap} \) defined as:
\[ c_{ap} = \tilde{z}_u^a, \forall a \in A_L \quad c_{ap} = \infty, \forall a = (v_l, t_v), v_l \in V_L. \]
By the max-flow min-cut theorem applied to \( G_L^{uw} \), it follows that for each \((u,v) \in K\), one can send one unit of flow from \( u_0 \) to \( t_v \) in \( G_L^{uw} \) using \( \tilde{z} \) (and hence \( \tilde{y} \) and \( \tilde{x} \))
as capacities. Since $f_{uv}^a \leq z_{va}^u$ holds for each $a \in A_L$ and $(u, v) \in K$, constraints (2e)–(2g) are implied by (3d)–(3f) which concludes the proof.

We also propose to extend the $(L_{CU})$ model by considering flow-balance constraints (4). For each source $u \in S$, they ensure that an outgoing arc of layered graph node $i_t$, $i \notin T^u \cup \{u\}$, can only be used in the arborescence associated to source $u$ if at least one incoming arc is chosen as well.

$$\sum_{a \in \delta^+(i)} z_{va}^u \leq \sum_{a \in \delta^-(i)} z_{va}^u \quad \forall u \in S, \forall i_t \in V_L : i \notin T^u \cup \{u\}$$

While the flow-balance constraints are not necessary to ensure validity of $(L_{CU})$, there exist cases in which they strengthen the associated LP relaxation. Figure 3 shows one of the rather typical situations in which this happens due to involved relay arcs. By adding flow-balance inequalities, the LP solution becomes integral and corresponds to the optimal solution shown in Figure 2. Observe that the two incoming arcs to node 5 belong to different variable sets in the corresponding LP solution to $(L_{MC})$ for which reason similar constraints are not strengthening there. Besides strengthening the LP relaxation of $(L_{CU})$, the flow-balance constraints also help to improve convergence by reducing the number of violated connectivity cuts (3b).

Figure 3: Optimal LP solution of $(L_{CU})$ on the layered graph for the input in Figure 1 without flow-balance constraints. Only arcs with associated non-zero variables are shown, labeled with the respective LP-values. The violation of flow-balance constraints at node 5 is marked bold and red.

2.3. Symmetry breaking constraints

By construction of the layered graph, it can sometimes happen that multiple feasible embeddings of rooted arborescences, one for each $u \in S$, exist. Each such embedding results in the same solution in the original graph, and hence, symmetries may be introduced in our $(L_{MC})$ and $(L_{CU})$ models. Since these symmetries may deteriorate the performance of branch-and-bound based approaches, we next introduce two families of symmetry breaking constraints. One typical situation arises if the routing paths of two different commodities contain a common node that needs to be used as a relay by only one of them.
Let \( i \in V \) be a node at which a relay has to be installed and let \( (u,v) \in K \) be a commodity that does not need to use \( i \) as a relay along its routing path. Assume that the distance to the previous relay (or the commodity source) of \( i \) along the path connecting \( u \) and \( v \) is equal to \( l \). Furthermore, let \((i,j)\) be the outgoing arc of node \( i \) on this path. Then, two feasible routing paths in \( G_L \) exist:

1. If the relay at \( i \) is not used, then layered graph arc \((i_{l_0}, j_{d_{ij}})\) belongs to the arborescence rooted at \( u \), and is used to connect \( u_0 \) to some copy of \( v \) in \( V_L \);
2. Alternatively, if the relay at \( i \) is used, the subpath given by \( \{(i_{l_0}, i), (i, j_{d_{ij}})\}\) is used instead.

To get rid of symmetries implied by such ambiguities, we force that in every feasible routing path installed relays are used whenever possible.

In case of (LCUT), this is enforced by constraints (5) that forbid the use of non-relay arcs emanating from some node \( i \), \( l > 0 \), if a relay is installed at node \( i \).

\[
\sum_{a = (i_{l_0}, j_{m}) \in A_L \setminus A_r} z^u_a \leq M^u_i \cdot (1 - y_i) \quad \forall u \in S, \forall i \in V, i \neq u 
\]

Thereby, \( M^u_i \) is a (tight) upper bound on the out-degree of node \( i \) in the arborescence rooted at \( u \in S \) which is defined as follows:

\[
M^u_i = \begin{cases} 
\min(|T^u|, |\delta^+(i)|), & \text{if } i \notin T^u \\
\min(|T^u| - 1, |\delta^+(i)|), & \text{otherwise}
\end{cases}
\]

For (LMCF) we use the stronger variant of the above symmetry breaking constraints

\[
\sum_{a = (i_{l_0}, j_{m}) \in A_L \setminus A_r} f^u_{a} \leq 1 - y_i \quad \forall (u, v) \in K, \forall i \in V, i \neq u 
\]

that exploit the fact that the binary flow variables are disaggregated per commodity. Thus, the outflow of each node is at most one.

### 3. Algorithmic framework

This section describes all implementation details that are relevant to ensure a good performance of our approaches. These include: (i) preprocessing techniques that aim to reduce the number of variables that have to be considered, (ii) further valid inequalities, (iii) the separation routines of all families of inequalities that are added dynamically, (iv) customized branching priorities, and (v) a heuristic to obtain initial solutions.

#### 3.1. Preprocessing

It is obvious, that a path connecting source \( u \in S \) to any target \( t \in T^u \) may not contain arcs that target node \( u \). Thus, for each commodity \((u, v) \in K \) and every arc \( a = (i_{l_0}, u_m) \in A_L \), we can remove the associated flow variable \( f^u_{a} \) in formulation (LMCF) and the layered arc variables \( z^u_a \) in formulation (LCUT).
Since a connection from \(u\) to \(v\) cannot use arcs emanating from \(v\), we can also remove flow variables \(f_{uv}^a\) of all arcs \(a = (v_l, j_m) \in A_L\) in model \((L_{MCF})\).

Further reductions may be possible for nodes \(i \in V\) whose in-degree or out-degree is equal to one at some layer. If \(\delta^-(i_l) = \{(j_p, i_l)\}\) for a node \(i_l \in V_L, i \notin S\), we can remove a possibly existing outgoing arc \((i_l, j_m)\) and all associated variables since using it would induce a cycle of length two in the original graph. Similarly, incoming arcs \((j_p, i_l)\) can be eliminated if \(\delta^+(i_l) = \{(i_l, j_m)\}\) in case \(i \notin T\). In case a non-source node becomes unreachable (i.e., all incoming arcs are removed), this node and all its outgoing arcs can be removed as well. Similarly, a non-target node without outgoing arcs can be removed together with all its incoming arcs. Additional reductions can be made if nodes are unreachable from a particular target, or have no remaining flow or layered arc variables associated to incoming (outgoing, respectively) arcs for a particular commodity or source node. In these cases, we eliminate the flow variables associated with that commodity or the layered arc variables associated with some source, respectively. These procedures are iteratively applied in several elimination rounds until no further reductions occur.

### 3.2. Valid inequalities

In this section, we describe two further families of valid inequalities for formulation \((L_{CUT})\). Though both are implied by the layered graph connectivity constraints \((3b)\) considering them before separating the latter inequalities typically turns out to be beneficial for the performance of our branch-and-cut approaches.

**Connectivity constraints on \(G\).** Connectivity constraints \((7)\) on the original graph are analogous to inequalities \((3b)\) on the layered graph.

\[
\sum_{a \in \delta^-(W)} x_a \geq 1 \quad \forall W \subset V : \exists (u, v) \in K, u \notin W, v \in W
\]

They ensure that each vertex set that separates source and target of a commodity must have at least one incoming arc. From the max-flow min-cut theorem, one can easily conclude that any solution satisfying constraints \((7)\) contains a path from \(u\) to \(v\) for every commodity \((u, v) \in K\). This path may, however, contain relay-free subpaths whose distance exceeds \(\lambda_{\text{max}}\). Thus, they are not sufficient to guarantee a feasible solution. A main advantage compared to the layered graph connectivity constraints \((3b)\) is that they are specified on the arc design variables of the original graph, thus each such cut influences all commodities. Since the number of connectivity constraints \((7)\) is exponential, we separate them dynamically; see Section 3.3 for details.

**Two-cycle inequalities.** Constraints \((8)\) ensure that an outgoing arc of some layered graph node can only be used if at least one incoming arc whose source is different from the outgoing arc’s target is used as well.

\[
\sum_{a' = (p, i_l) \in \delta^-(i_l); p \neq j} z_{a'}^u \geq z_u^u \quad \forall u \in S, \forall a = (i_l, j_m) \in A_L : i \neq u
\]
Two-cycle inequalities (8) are implied by layered graph connectivity constraints (3b) and do not strengthen the formulation [5]. Similar to cut constraints (7), they are, however, beneficial for reducing the number of dynamically separated cut-set inequalities (3b).

Since the number of flow-balance constraints (4) and two-cycle inequalities (8) is pseudo-polynomial, two implementation variants are considered in our computations: (i) adding them exhaustively to the initial formulation, and (ii) separating them dynamically. Details of the used separation procedures are given below in Section 3.3.

3.3. Separation

In this section we describe the separation procedure used in our branch-and-cut approach for formulation (L\text{CUT}) that dynamically adds layered connectivity constraints (3b), flow balance constraints (4), connectivity constraints on the original graph (7), and two-cycle elimination constraints (8).

First, possibly violated cut-set constraints on the original graph are identified using the maximum flow algorithm by Cherkassky and Goldberg [3]. Thereby, so-called nested cuts (see, e.g., Ljubić et al. [18]) are considered which means that the capacities of all arcs included in just added cuts are set to one and the flow computation is subsequently repeated to possibly find further violated inequalities. This procedure is applied for each commodity pair until no further violated inequalities are found. Before proceeding with the next commodity pair, all arc capacities are reset (to the original values induced by the current LP solution). Since this procedure may yield identical cuts for different commodity pairs, we employ a duplicate detection and stop separating cuts for the current commodity as soon as a duplicate is identified. To avoid introducing an unwanted bias due to the order in which the commodities are processed, we consider them in a random order based on a fixed-seed value.

Once this first separation routine terminates, we add violated flow-balance (4) and two-cycle inequalities (8). Separation of these constraints is performed by inspecting the LP values of relevant variables for all not yet added inequalities. Complete separation of the two-cycles turned out to be too inefficient. Therefore, we resort to a slightly simpler approach that adds those inequalities if the following condition is violated.

\[
\sum_{a' \in \delta^-(u)} z^u_{a'} \geq z^u_a \quad \forall u \in S, \forall a = (i_l, j_m) \in A_L : i \neq u
\]

Finally, we separate layered graph connectivity cuts (3b) in case neither flow-balance nor two-cycle inequalities have been added in the previous step. Such a conditional separation is beneficial to avoid adding too many redundant constraints. As before, violated cuts are identified by maximum flow computations using the algorithm from [3]. To detect a violated inequality of type (3b), for each \((u, v) \in K\), we construct the layered graph \(G^u_{lv}\) as described in the proof of Theorem 2, and calculate the maximum flow between \(u_0\) and the target node \(t_v\), using \(z^u_a\) values as arc capacities for the arcs from \(A_L\) and \(\infty\) for the arcs adjacent to \(t_v\). If the obtained flow is less than one, the violated cut is added to the model. Again, we consider nested cuts, duplicate handling, and fixed-seed randomization for the order in which the commodity pairs are processed.
1 separate cut-set inequalities (7) on the original graph
2 separate flow-balance constraints (4)
3 separate two-cycle inequalities (8)
4 if no flow-balance constraints and two-cycle inequalities added then
5 separate cut-set inequalities (3b) on the layered graph

Algorithm 3.1: Separation Procedure

The separation procedure is outlined in Algorithm 3.1.

In the second implementation variant of our branch-and-cut (B&C) approach, in which flow-balance constraints and two-cycle inequalities are added in the initialization phase, the layered cut-set inequalities are separated unconditionally.

To avoid separating too many inequalities, we only add cut-set inequalities if they are violated by a value of at least 0.5. Flow-balance constraints and two-cycle inequalities, however, are separated without such a threshold.

3.4. Branching

Several properties of feasible solutions are enforced on the layered graph. The objective function, however, depends solely on the variables corresponding to the original graph. Moreover, decisions concerning the arcs available in the original graph directly influence the layered graph variables. Hence, it is reasonable to focus on the former for branching decisions. Regarding the two types of variables for the original graph—edge and relay variables—it is natural to prioritize the relay variables since placing a relay is usually much more expensive than installing a connection along an arc. Thus, our algorithms use a cost-based branching priority in which the priority of each variable is equivalent to its cost.

To stay consistent with the literature we do not use custom branching priorities for the node-arc formulation.

3.5. Initial heuristic

Before starting the B&C algorithm, we compute a feasible solution and hand it over to the MILP solver as initial primal bound. The heuristic resembles Prim’s algorithm for spanning trees: it computes optimal paths for one commodity at a time and sets the costs of used arcs and relays to zero before it proceeds with the next source-target pair. When only a single commodity pair is given the DNDPR is known as the minimum cost path problem with relays (MCPPR). This latter problem can be solved exactly using an efficient dynamic programming (DP) algorithm proposed by Laporte and Pascoal [12]. However, different to the DNDPR, the MCPPR also allows connecting commodities by non-simple paths. We therefore adjust the DP algorithm from Laporte and Pascoal [12] by keeping track of already visited nodes in each state, in order to disallow extensions that form cycles.

To improve upon the basic algorithm, we consider some extensions. Observe that the design of the heuristic entails a strong influence of the order in which the commodities are processed. We attempt to reduce this influence by considering ten different permutations based on fixed-seed randomization and then keep the best solution.

Additionally, within the DP algorithm for the MCPPR we not only order the labels by non-decreasing cost but also break ties by favoring paths that
reach the considered node at smaller distance. Once the best solution among the ten runs has been identified, we run the DP algorithm once again with the costs of all used relays and arcs set to zero. In certain cases this helps to avoid redundancies resulting in a smaller cost. We note that similar heuristic ideas based on sequential upgrade of a partial solution have been successfully used in [13, 14].

4. Computational study

In this section we present computational results for the considered algorithms and variants. We start by giving details on the computational environment, used test instances and the motivation for their selection. Finally, we present the obtained results.

Our algorithms are implemented in C++ using CPLEX 12.7.1 as a general-purpose MILP solver. All experiments have been performed in single thread mode with default parameter settings. Experiments have been executed on an Intel Xeon E5-2670v2 machine with 2.5 GHz. The computation time limit has been set to 7 200 seconds.

In the following we compare the four solution approaches described in Table 1. Note that in both $\text{LCUT-s}$ and $\text{LCUT-d}$, cut-set inequalities (3b) and (7) are separated dynamically. In $\text{LCUT-s}$, flow-balance constraints (4) and two-cycle inequalities (8) are added initially to the model while in $\text{LCUT-d}$ these two sets are separated dynamically as described in Section 3.3.

4.1. Instances

Benchmark instances used by the authors of [14] are no longer available (personal communication with X. Li). Due to the lack of other existing benchmark instances for the DNDPR we constructed new ones based on existing instances for the NDPR. We consider two groups of benchmark instances: i) asymmetric instances derived from Cabral et al. [1] and ii) symmetric instances derived from Konak [10]. In the following we shortly outline the construction procedures for the original instances which involve undirected graphs, and then discuss our adjustments to obtain directed graphs. These instances will be made available after the completion of the reviewing process.

Cabral instances. Cabral et al. [1] generated instances in which the underlying graph is a square grid graph (i.e., each node is connected to its direct vertical and horizontal neighbors). Integral edge costs and distances are chosen uniformly at random from the interval [10,30] and the distance limit $\lambda_{\text{max}}$ is equal to 70. Nine square grid graphs with $A$ rows and $B$ columns (i.e., with $|V| = A \cdot B$ nodes) have been generated using $(A,B) \in \{(4,5),(5,5),(6,5),(7,5),(8,5),(9,5),(10,5),(11,5),(12,5)\}$. A total of 180 instances has been generated through creating 10 instances (by random sampling
of the commodities) for each graph and each considered number of commodities $|K| \in \{5,10\}$. In particular, all commodities of each instance have the same source node, i.e., $|S| = 1$. We remark that duplicate commodities exist in some of the instances from Cabral et al. [1] which can be removed in a preprocessing step.

We obtained directed instances by replacing each edge by two directed arcs. Distance and cost of the first arc are equivalent to those of the edge. The values for the second arc are chosen uniformly at random from the interval $[10,30]$. Instances for which the specified number of commodities does not match the actual number have been corrected by inserting sufficiently many new commodities and preserving the property that $|S| = 1$. Note that [14] uses the same approach to construct their instances although they use new base graphs instead of those from [1]. Thus, our new instance set is comparable in size and structure. The basic instance properties ($|V|$, $|E|$, $|K|$, and $\lambda_{\text{max}}$) are shown in Table 3.

**Konak instances.** Konak [10] generated instances by first placing $|V| \in \{40,50,60,80,160\}$ nodes at random integer coordinates $(x,y) \in [0,100] \times [0,100]$. Edges are added randomly such that $|E| = 198$ (for $|V| = 40$) and $|E| = 3624$ (for $|V| = 160$). Their length is set to the Euclidean distance $d_{ij}$ between the two adjacent nodes $i$ and $j$, while the costs are either equal to the edge length (type I) or equal to $\lambda_{\text{max}} - d_{ij}$ (type II). Using $\lambda_{\text{max}} \in \{30,35\}$ and $|K| \in \{5,10\}$, 20 instances have been generated for each of the two types. Each of these instances typically contains multiple sources and targets.

We adapt these instances by rounding up the edge distances and costs to the nearest integer values. Each edge is replaced by two directed arcs. However, this time both arcs have the cost and distance of the original edge. This is done to keep the instance Euclidean and also to preserve the direct or indirect correlation of edge costs and distances. See Table 4 for an overview.

### 4.2. Comparison to the state-of-the-art

As indicated above, we could not obtain the instances used in the previous literature. However, the Cabral instances are comparable in structure and size to those tested in [14], which allows us to use them to obtain at least an intuition on how our exact algorithms compare to those presented in [14]. As reference point we employ the node-arc formulation. We compare speedups between NA and the best B&P approach from the literature as well as the algorithms based on our layered graph formulations. The respective values are computed by $t_{\text{NA}}/t_{\text{alg}}$ for alg \in \{B&P, L\_MCF, L\_CUT-s, L\_CUT-d\}. Since the Cabral instances feature 10 instances of each type, results have been aggregated by computing averages. The results are shown in Table 2. The speedup values of our algorithms are similar to those from the literature for the smallest instances but they are considerably better for the larger ones. Also note that on our instances the node-arc formulation sometimes terminated prematurely due to the time limit. Allowing the algorithm to finish—as done in [14]—would have resulted in even larger speedups.

Without the original benchmark set, a precise comparison is impossible. Yet these results indicate that our algorithms are at least as fast as those from the existing literature and most likely outperform them significantly.
Table 2: Speedup ratio to the node-arc formulation. Values have been obtained by dividing the computation time of the node-arc formulation through the computation time of the respective algorithm. The first column has been obtained by extracting the respective results from [14]. The highest speedup per column is marked bold.

<table>
<thead>
<tr>
<th>Instance</th>
<th>B&amp;F (Li et al.)</th>
<th>L\text{MCF}</th>
<th>L\text{CUT-s}</th>
<th>L\text{CUT-d}</th>
</tr>
</thead>
<tbody>
<tr>
<td>04A05B70L05K</td>
<td>8.4</td>
<td>6.2</td>
<td>3.1</td>
<td>3.2</td>
</tr>
<tr>
<td>04A05B70L10K</td>
<td>55.9</td>
<td>45.2</td>
<td>35.7</td>
<td>19.8</td>
</tr>
<tr>
<td>05A05B70L05K</td>
<td>20.8</td>
<td>21.0</td>
<td>15.7</td>
<td>14.1</td>
</tr>
<tr>
<td>05A05B70L10K</td>
<td>107.3</td>
<td>66.9</td>
<td>53.6</td>
<td>44.9</td>
</tr>
<tr>
<td>06A05B70L05K</td>
<td>7.1</td>
<td>40.2</td>
<td>25.4</td>
<td>22.3</td>
</tr>
<tr>
<td>06A05B70L10K</td>
<td>61.1</td>
<td>134.6</td>
<td>104.4</td>
<td>98.7</td>
</tr>
<tr>
<td>07A05B70L05K</td>
<td>31.8</td>
<td>45.6</td>
<td>19.2</td>
<td>16.2</td>
</tr>
<tr>
<td>07A05B70L10K</td>
<td>34.6</td>
<td>321.2</td>
<td>476.3</td>
<td>289.8</td>
</tr>
<tr>
<td>08A05B70L05K</td>
<td>9.3</td>
<td>454.4</td>
<td>216.3</td>
<td>81.7</td>
</tr>
<tr>
<td>08A05B70L10K</td>
<td>92.0</td>
<td>466.2</td>
<td>543.7</td>
<td>218.6</td>
</tr>
<tr>
<td>09A05B70L05K</td>
<td>9.9</td>
<td>225.4</td>
<td>110.9</td>
<td>69.0</td>
</tr>
<tr>
<td>09A05B70L10K</td>
<td>40.5</td>
<td>256.8</td>
<td>301.6</td>
<td>237.8</td>
</tr>
<tr>
<td>10A05B70L05K</td>
<td>49.9</td>
<td>631.0</td>
<td>319.8</td>
<td>122.1</td>
</tr>
<tr>
<td>10A05B70L10K</td>
<td>33.6</td>
<td>876.1</td>
<td>1337.3</td>
<td>683.9</td>
</tr>
<tr>
<td>11A05B70L05K</td>
<td>25.1</td>
<td>540.6</td>
<td>306.5</td>
<td>123.0</td>
</tr>
<tr>
<td>11A05B70L10K</td>
<td>45.4</td>
<td>954.9</td>
<td>1500.4</td>
<td>555.7</td>
</tr>
<tr>
<td>12A05B70L05K</td>
<td>5.2</td>
<td>716.9</td>
<td>528.4</td>
<td>215.0</td>
</tr>
<tr>
<td>12A05B70L10K</td>
<td>110.1</td>
<td>755.3</td>
<td>1164.3</td>
<td>405.9</td>
</tr>
</tbody>
</table>

4.3. LP relaxation bounds

In the following, we compare the quality of lower bounds that can be obtained by the three algorithms from Table 1: L\text{MCF}, L\text{CUT} (note that both L\text{CUT-s} and L\text{CUT-d} provide the same lower bounds), and NA. When computing LP bounds we deactivate CPLEX presolving, general purpose heuristics, and general purpose cuts. In addition, no threshold value is set for the separation of cut-set inequalities (3b) and (7).

LP gaps are computed as $(UB^* - LB)/UB^*$ where $UB^*$ is the best known upper bound and $LB$ is the lower bound obtained by the LP relaxation. Table 3 reports results obtained on the Cabral instances, and Table 4 provides results for the Konak instances.

**Cabral instances.** We observe that the algorithms based on (L\text{CUT}) yield the strongest bounds. L\text{MCF} follows closely behind but always delivers strictly weaker bounds. The reason for this is the fact that the Cabral instances consider only a single source node. This means that algorithms based on (L\text{CUT}) use precisely one set of variables with respect to the layered graph on which they model an arborescence. The multi-commodity flow formulation, on the other hand, uses one set of variables per commodity pair. In this situation the cut model benefits from aggregating per source which enables it to obtain a stronger bound by means of flow-balance constraints (4). Moreover, the cut formulation yields a much smaller model here with respect to the number of variables. In general, both algorithms based on layered graph models deliver excellent bounds well below the one percent mark. As expected, the node-arc formulation is the weakest model with significantly worse bounds than the layered graph models.

**Konak Instances.** Compared to the Cabral instances we face much denser graphs here. Moreover, we are now dealing with multiple source nodes in-
Table 3: Results for the LP bounds on the directed Cabral instances. Each line represents the average across ten instances. Per row the strongest bounds are marked bold.

| Instance | $|V|$ | $|E|$ | $\lambda_{\text{max}}$ | $|K|$ | Properties | LP gap [%] | $L_{\text{MCF}}$ | $L_{\text{CUT}}$ | NA |
|----------|-----|-----|----------------|-----|----------|-------------|---------|----------------|-----|
| 04A05B70L05K | 20 | 62 | 70 | 5 | | 0.2 | 0.0 | 27.6 |
| 04A05B70L10K | 20 | 62 | 70 | 10 | | 0.2 | 0.0 | 35.0 |
| 05A05B70L05K | 25 | 80 | 70 | 5 | | 0.8 | 0.0 | 31.4 |
| 05A05B70L10K | 25 | 80 | 70 | 10 | | 0.1 | 0.0 | 34.4 |
| 06A05B70L05K | 30 | 98 | 70 | 5 | | 0.5 | 0.0 | 36.8 |
| 06A05B70L10K | 30 | 98 | 70 | 10 | | 0.6 | 0.0 | 34.9 |
| 07A05B70L05K | 35 | 116 | 70 | 5 | | 0.1 | 0.0 | 40.5 |
| 07A05B70L10K | 35 | 116 | 70 | 10 | | 0.7 | 0.1 | 40.6 |
| 08A05B70L05K | 40 | 134 | 70 | 5 | | 0.1 | 0.0 | 45.1 |
| 08A05B70L10K | 40 | 134 | 70 | 10 | | 1.0 | 0.1 | 40.2 |
| 09A05B70L05K | 45 | 152 | 70 | 5 | | 0.1 | 0.0 | 42.9 |
| 09A05B70L10K | 45 | 152 | 70 | 10 | | 0.7 | 0.0 | 39.8 |
| 10A05B70L05K | 50 | 170 | 70 | 5 | | 0.1 | 0.0 | 46.2 |
| 10A05B70L10K | 50 | 170 | 70 | 10 | | 0.9 | 0.0 | 43.9 |
| 11A05B70L05K | 55 | 188 | 70 | 5 | | 0.5 | 0.0 | 46.2 |
| 11A05B70L10K | 55 | 188 | 70 | 10 | | 0.2 | 0.1 | 42.5 |
| 12A05B70L05K | 60 | 206 | 70 | 5 | | 0.5 | 0.1 | 43.3 |
| 12A05B70L10K | 60 | 206 | 70 | 10 | | 0.8 | 0.1 | 42.6 |

Instead of just a single one. This means that now also the ($L_{\text{CUT}}$) formulation requires multiple sets of layered graph variables. Under these circumstances we still obtain strong LP bounds but not as strong as on the Cabral instances, see Table 4. For the larger instances of type I it becomes challenging to solve the LP relaxation to optimality indicated by dashes in the table. The instances with indirectly correlated costs (type II) turned out to be much easier to solve. Here $L_{\text{MCF}}$ as well as $L_{\text{CUT}}$ provide results for all instances before exceeding the time limit. The results indicate that the bound strength is excellent if the computations can be completed. As before, we observe that the bounds provided by $L_{\text{CUT}}$ are at least as strong as those of $L_{\text{MCF}}$. The node-arc formulation (NA) yields much weaker bounds but also terminates significantly faster. Therefore, NA gives the only bounds for the two largest instances of type I where the time limit is reached for the layered graph models. In contrast to the layered graph models, the node-arc formulation seems to work much better on type I instances than on type-II instances where the bounds are much worse, roughly by a factor of two.

4.4. Overall performance

Cabral instances. Our layered graph models are able to solve all 180 instances to proven optimality. The MILP runs are consistent with the LP results, however, $L_{\text{MCF}}$ is now much closer to the cut model despite its worse bounds, see Table 5. Optimality gaps are computed by $(UB^* - LB)/UB^*$ where $UB^*$ is the best known upper bound and $LB$ is the lower bound obtained by the investigated algorithm.

We observe a clear difference between the static and the dynamic variant of the cut formulation $L_{\text{CUT-s}}$ and $L_{\text{CUT-d}}$, respectively. The reasons for the advantage of the static approach are the sparseness and the size of the input graphs. Both lead to rather small models and the overhead for adding the valid inequalities in advance is manageable. Therefore, the slowdown for
Table 4: Results for the LP bounds on the directed Konak instances. Missing gap values correspond to runs that did not complete within the time limit. Bold values indicate the tightest bounds per type and instance.

| Instance | $|V|$ | $|E|$ | $\lambda_{\text{max}}$ | $|K|$ | LP gap [%] type I | LP gap [%] type II |
|----------|------|------|-----------------|------|----------------|-----------------|
| 040N_05K_30L | 40 | 396 | 30 | 5 | 5.2 | 0.0 | 0.0 | 75.1 |
| 040N_05K_35L | 40 | 544 | 35 | 5 | 5.6 | 0.4 | 0.4 | 70.2 |
| 040N_10K_30L | 40 | 396 | 30 | 10 | 7.8 | 4.0 | 4.0 | 74.0 |
| 040N_10K_35L | 40 | 544 | 35 | 10 | 5.6 | 4.6 | 4.6 | 68.5 |
| 050N_05K_30L | 50 | 558 | 30 | 5 | 1.3 | 0.0 | 0.0 | 71.7 |
| 050N_05K_35L | 50 | 744 | 35 | 5 | 0.0 | 0.0 | 0.0 | 80.1 |
| 050N_10K_30L | 50 | 558 | 30 | 10 | 12.2 | 8.6 | 8.6 | 76.0 |
| 050N_10K_35L | 50 | 744 | 35 | 10 | 12.0 | 9.0 | 9.0 | 80.0 |
| 060N_05K_30L | 60 | 610 | 30 | 5 | 7.5 | 6.3 | 6.3 | 82.8 |
| 060N_05K_35L | 60 | 824 | 35 | 5 | 0.0 | 0.0 | 0.0 | 75.0 |
| 060N_10K_30L | 60 | 610 | 30 | 10 | 12.9 | 12.9 | 12.9 | 79.7 |
| 060N_10K_35L | 60 | 824 | 35 | 10 | 3.6 | 3.6 | 3.6 | 74.7 |
| 080N_05K_30L | 80 | 1282 | 30 | 5 | 0.0 | 0.0 | 0.0 | 74.7 |
| 080N_05K_35L | 80 | 1706 | 35 | 5 | 0.3 | 0.3 | 0.3 | 70.7 |
| 080N_10K_30L | 80 | 1282 | 30 | 10 | 3.2 | - | - | 61.8 |
| 080N_10K_35L | 80 | 1706 | 35 | 10 | - | - | - | 69.6 |
| 160N_06K_30L | 160 | 5546 | 30 | 5 | 0.0 | 0.0 | 0.0 | 77.2 |
| 160N_06K_35L | 160 | 7248 | 35 | 5 | 0.5 | 0.5 | 0.5 | 71.2 |
| 160N_10K_30L | 160 | 5546 | 30 | 10 | - | - | - | 71.5 |
| 160N_10K_35L | 160 | 7248 | 35 | 10 | - | - | - | 61.7 |

Solving the LP relaxations is negligible but we can reduce the number of cut iterations in each node of the B&C tree significantly. Similarly, we also observe that much fewer branch-and-bound (B&B) nodes—about 17% on average—are needed until optimality can be proven. $L_{\text{MCF}}$, however, performs better in this respect. It solves the majority of instances already at the root node and the few “outliers”—18 in total—with at most 19 B&B nodes. The cut formulation, albeit being stronger, solves 35 instances fewer at the root node and requires up to 67 B&B nodes when adding flow-balance and two-cycle inequalities statically. The reasons for this seem to be that the fractional solutions of $L_{\text{MCF}}$ are closer to being feasible and that $L_{\text{MCF}}$ interacts better with the solver since no further inequalities are added in the solution process. Although the computation times of $L_{\text{MCF}}$ and $L_{\text{CUT}}$ are quite similar, we still observe that the former is considerably more sensitive to changes in the number of commodities due to the resulting increase in model size, see also Table 2. The node-arc formulation is significantly outperformed and cannot even solve all instances to optimality within the time limit.

Konak instances. The results of solving the MILP formulations are provided in Tables 6 and 7. In accordance with the experiments on the LP bounds, type II instances are again easier to solve than type I instances. $L_{\text{MCF}}$ and $L_{\text{CUT}}$-d solve all instances of type II to optimality. $L_{\text{CUT}}$-s, on the other hand, cannot solve the largest two instances of this set to optimality. Similarly, $L_{\text{CUT}}$-s solves fewer instances of type I to optimality than the dynamic variant and provides larger gaps whenever both terminate prematurely due to the time limit. As the graphs are denser here than the 4-grid graphs of the Cabral instances, it is no longer beneficial to add all valid inequalities in advance. Thus, dynamic
Table 5: Results for the directed Cabral instances. Each line represents the average across ten instances. Column #opt provides the number of optimally solved instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>gap [%]</th>
<th>time [s]</th>
<th>#opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_MCF</td>
<td>L_CUT-s</td>
<td>L_CUT-d</td>
<td>NA</td>
</tr>
<tr>
<td>NA</td>
<td>L_MCF</td>
<td>L_CUT-s</td>
<td>L_CUT-d</td>
</tr>
<tr>
<td>04A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>04A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>05A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>05A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>06A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>06A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>07A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>07A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>08A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>08A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>09A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>09A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>10A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>10A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>11A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>11A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>12A05B70L05K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>12A05B70L10K</td>
<td>0.0</td>
<td>0.0</td>
<td>&lt; 1</td>
</tr>
</tbody>
</table>

separation helps to reduce the size of the LP relaxations. While being slightly slower on some type II instances, L_MCF outperforms the L_CUT approaches on the type I instances. There it solves three more instances to optimality and features considerably smaller computation times on the remaining ones. However, for the largest two instances it fails to provide any non-trivial bounds. L_CUT-d terminates still in the root note, but at least provides reasonable bounds. It is noticeable that L_MCF proves optimality for 15 out of 20 type II instances and 5 out of 20 type I instances already at the root node, including several cases with non-zero LP gap. L_CUT-s performs quite similar in this respect and solves 3 type I and 14 type II instances at the root node. L_CUT-d, on the other hand, achieves this only for a single type II instance. Again, we presume that L_MCF interacts better with the solver due to having all information available from the beginning. The node-arc formulation is not competitive and features large gaps even on the smaller instances. In contrast to the results of the LP runs it provides better results on the type II instances, too.

5. Conclusion

We introduced two exact solution approaches for the directed network design problem with relays (DNDPR) based on layered graphs. The first approach relies on a multi-commodity flow formulation (L_MCF) which is pseudo-polynomial in size, and the second approach is a branch-and-cut (B&C) approach based on the (L_CUT) model with an exponential number of constraints. Both models provide extremely tight linear programming (LP) bounds on the considered benchmark instances. We proposed additional valid inequalities for strengthening the (L_CUT) formulation and also for breaking symmetries induced by the layered graph structure. We investigated two approaches for adding the strengthening inequalities to the (L_CUT) formulation: a static and a dynamic one. For the instances from Cabral et al. [1] representing sparse 4-grid graphs
### Table 6: Results for the directed Konak instances of type I.

<table>
<thead>
<tr>
<th>Instance</th>
<th>gap [%]</th>
<th>time [s]</th>
<th>L_MCF</th>
<th>L_CUT-s</th>
<th>L_CUT-d</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>040N05K30L</td>
<td>0.0</td>
<td>10.0</td>
<td>46.1</td>
<td>0.0</td>
<td>68.4</td>
<td>2022</td>
</tr>
<tr>
<td>040N05K35L</td>
<td>0.0</td>
<td>10.0</td>
<td>50.8</td>
<td>0.0</td>
<td>62.5</td>
<td>2022</td>
</tr>
</tbody>
</table>

### Table 7: Results for the directed Konak instances of type II.

<table>
<thead>
<tr>
<th>Instance</th>
<th>gap [%]</th>
<th>time [s]</th>
<th>L_MCF</th>
<th>L_CUT-s</th>
<th>L_CUT-d</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>040N05K30L</td>
<td>0.0</td>
<td>10.0</td>
<td>46.1</td>
<td>0.0</td>
<td>68.4</td>
<td>2022</td>
</tr>
<tr>
<td>040N05K35L</td>
<td>0.0</td>
<td>10.0</td>
<td>50.8</td>
<td>0.0</td>
<td>62.5</td>
<td>2022</td>
</tr>
</tbody>
</table>
with very few commodities, it turns out that the overhead of adding more inequalities in the initialization phase is negligible. On the contrary, for larger and denser instances from Konak [10], this overhead does not pay off, and dynamic separation of strengthening inequalities is recommended.

The overall performance of the proposed layered graph approaches based on \((L_{MCF})\) and \((L_{CUT})\) is comparable. In general, we observed that the former performs slightly better on sparse graphs with very few commodities, whereas the latter can be used as an alternative for denser graphs with larger number of commodities, when memory issues with the general-purpose solvers are faced.

Using the existing node-arc formulation as base line for a comparison with the existing approaches, we showed that our exact approaches are significantly faster than the state-of-the-art from [14].


