Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Decomposition Methods for Large-Scale Network Expansion Problems

Ioannis Fragkos
Rotterdam School of Management, The Netherlands, fragkos@rsm.nl
Jean-François Cordeau, Raf Jans
HEC Montréal and CIRRELT, Canada, jean-francois.cordeau@hec.ca, raf.jans@hec.ca

Network expansion problems are a special class of multi-period network design problems in which arcs can be opened gradually in different time periods but can never be closed. Motivated by practical applications, we focus on cases where demand between origin-destination pairs expands over a discrete time horizon. Arc opening decisions are taken in every period, and once an arc is opened it can be used throughout the remaining horizon to route several commodities. Our model captures a key timing trade-off: the earlier an arc is opened, the more periods it can be used for, but its fixed cost is higher, since it accounts not only for construction but also for maintenance over the remaining horizon. For the capacitated variant, we develop an arc-based Lagrange relaxation, combined with local improvement heuristics. For uncapacitated problems, we develop four Benders decomposition formulations and show how taking advantage of the problem structure leads to enhanced algorithmic performance. We then utilize real-world and artificial networks to generate 1,080 instances, with which we conduct a computational study. Our results demonstrate the efficiency of our algorithms. Notably, for uncapacitated problems we are able to solve instances with 2.5 million variables to optimality in less than two hours of computing time.

Key words: Network expansion, Lagrange relaxation, Benders decomposition, Heuristics

1. Introduction.

Network expansion models represent a variety of problems arising in fields as diverse as road construction (Yang et al. 1998), logistics (Lee and Dong 2008), energy transport and telecommunications (Minoux 1989), and railways (Hooghiemstra et al. 1999). These problems exhibit a multi-period structure: given a planning horizon and a demand forecast therein, one needs to decide which arcs to open and when, so that the resulting network can accommodate the typically increasing demand throughout the planning horizon. Concretely, opening an arc implies that this arc can be used to route commodities until the end of the planning horizon. However, the more periods the arc is under operation, the higher the total fixed cost that is incurred. This fixed opening cost represents the total expense over the remaining horizon, such as construction and maintenance costs for building a track in a rail network.

In this paper, we study an archetypal formulation that captures the key timing trade-off of network expansion decisions. On the one hand, building an arc early on implies a high fixed cost, but the arc can be used to route commodities for a large number of subsequent periods. On the other hand, building an arc later in the horizon is associated with a lower fixed cost, but the number of periods in which the arc can be used is smaller. The objective is then to jointly minimize the arc construction and operating costs over the given planning horizon.

Although such network expansion formulations can provide useful input for strategic and tactical decisions, their very large scale makes them difficult or even impossible to solve with modern mixed-integer programming (MIP) technology. To this end, we exploit their multi-period structure to devise specialized decomposition algorithms for both capacitated and uncapacitated variants. First, we apply arc-based Lagrange relaxation for the capacitated problem. Second, we develop a stand-alone heuristic which we combine with Lagrange relaxation. Third, we apply

Benders decomposition to uncapacitated problems, by decomposing the original problem into single-period shortest path subproblems per period and per commodity. We then show how Pareto-optimal Benders cuts can be generated efficiently for our application, and compare the resulting implementations with the novel formulation of Fischetti et al. (2010).

In order to illustrate the computational efficiency of our algorithms, we generate new instances. Specifically, we utilize three actual shipping networks that were used in Pazour et al. (2010) to investigate the usefulness of designing a high-speed freight rail network in the United States, and a subset of the `R` instances (Crainic et al. 2001), which have been used in several other studies (Katayama et al. 2009, Yaghini et al. 2014, Costa et al. 2009). In total, we construct more than a thousand instances, with which we perform extensive computational experiments. We show that our heuristics, Lagrange relaxation and Benders decomposition are efficient in finding high-quality solutions within a reasonable amount of time, while their performance scales well for larger problem instances. Notably, we are able to solve to optimality instances with 2.5 million variables in less than two hours of CPU time.

The remainder of the paper is organized as follows. We first review related research in Section 2. Then, Section 3 describes the problem formulation and Section 4 explains the construction of a heuristic tailored to large-scale instances. Section 5 provides details on the Lagrange relaxation and Benders decomposition. Then, Section 6 presents computational results. We conclude by reflecting on future research avenues, reported in Section 7.

2. Literature Review.

The literature on network design and expansion problems is voluminous. In what follows, we first focus on applications related to capacitated and uncapacitated multi-period problems and then provide an overview of methodological advancements in the larger field of network design problems.

2.1. Applications of Multi-period Problems.

Bärmann et al. (2017) consider the problem of expanding the German railway network, whose demand is anticipated to increase by 50% in the coming two decades. Their model captures capacity installation decisions that span over multiple periods. In their setting, investments have to be paid throughout the construction periods but the corresponding capacity becomes available only at the last construction period. Related to our model is also the work of Papadimitriou and Fortz (2015), who study multi-period networks with capacity installation, link maintenance and routing decisions in telecommunication networks. Their work is similar to the problem studied here but incorporates features specific to telecommunication networks, such as link aging, non-linear routing cost and switching capacity.

Another related problem is the stochastic network expansion problem for railway capacity planning, considered in Lai and Shih (2013), where the authors minimize a combination of network upgrading costs, expected arc operations costs and unfulfilled demand. Other research, such as Blanco et al. (2011) and Marin and Jaramillo (2008), focuses on application-specific transportation network expansion models using heuristics, while Petersen and Taylor (2001) develop a decision support system to investigate the economic viability of a new railway in Brazil. Cheung et al. (2001) highlight the importance of long-term planning by noting that "Today's optimal service network may not be adequate in the future [...] Therefore, we needed to base the network design on a long-term demand forecast". The authors study the problem of redesigning DHL's network of depots and service centers, determining facility capacities and opening timing by solving a multi-period facility location problem, which can be recast as a network design problem, with each facility converted to an arc. In a similar fashion, Pazour et al. (2010), who consider the design of a high-speed rail network for cargo distribution, note that such a network is likely to be built across multiple periods, and assume an incremental design plan

by restricting the total length of the network and by fixing prior line construction decisions.

Systems with ample capacity or systems where the pertinent decision is determining the network topology can be represented by uncapacitated models. Capacity restrictions are relevant for freight rail networks, but the study of uncapacitated models can be useful since capacity restrictions can be tackled by post-processing strategies, as in Pazour et al. (2010), who assume that excess flows are directed from high speed rail to the road network. Yet another interesting application is the infrastructure expansion problem in the coal export supply chain, studied by Kalinowski et al. (2015). The authors consider an uncapacitated multi-period network design problem which can be described by our uncapacitated model formulation. There, our model arises when the authors devise an alternative formulation, which results in a model with a multi-period structure, where the number of periods equals the difference between the ultimate and the initial flow. A special characteristic of this formulation is that it has a single commodity whose demand increases by one unit each time period. The same group (Baxter et al. 2014) has studied a single-commodity incremental network design problem with shortest paths, in which they are able to characterize the structure of optimal solutions and derive a 4-approximation algorithm. Such papers are representative of incremental optimization, a research stream pioneered by Şeref et al. (2009). Our paper adds to this literature by designing tailored algorithms for multi-commodity variants of such incremental problems.

2.2. Methodology.

In a seminal paper, Balakrishnan et al. (1989) utilize dual ascent methods to solve singe-period uncapacitated network design problems with up to 500 integer and 1.98 million continuous variables. Subsequent works focus mostly on exact approaches,

such as Lagrange relaxation-based branch-and-bound (Cruz et al. 1998, Holmberg and Yuan 1998), branch-and-cut, and Benders decomposition (Randazzo and Luna 2001, Rahmaniani et al. 2016). To the best of our knowledge, the Lagrange relaxation algorithm of Holmberg and Yuan (1998) is the state-of-the-art exact approach for solving single-period uncapacitated problems. A heuristic used by several researchers is the dynamic slope scaling approach of Kim and Pardalos (1999), which the authors utilized for single-commodity uncapacitated and capacitated network design problems. The main idea of slope scaling is to update the objective function coefficients of the continuous variables dynamically, so that the adjusted linear cost is a locally good approximation of both the linear and fixed costs. This idea was adopted by other authors for problems with richer structures, such as capacitated network design (Crainic et al. 2004) and freight rail transportation (Zhu et al. 2014).

In spite of some notable research output on the single-period uncapacitated problem, the largest literature stream has focused on solution methods for its capacitated counterpart. To this end, some authors have conducted polyhedral studies (Atamtürk 2002, Atamtürk and Rajan 2002, Bienstock and Günlük 1996, Raack et al. 2011, Günlük 1999), while others have used column generation (Frangioni and Gendron 2009, 2013) and Lagrange relaxation (Frangioni and Gorgone 2014). In terms of other exact methods, we note Chouman et al. (2017), who develop a cutting-plane algorithm which utilizes five classes of valid inequalities.

Since single-period network design is computationally challenging, most authors have adopted heuristic approaches. Yaghini et al. (2014) use a tabu-search algorithm with a neighborhood induced by families of valid inequalities, while Paraskevopoulos et al. (2016) use scatter and local search alongside new search operators that allow partial rerouting of multiple commodities. The computational

experiments show that those two approaches are perhaps the most efficient heuristics at the time of this writing, while Katayama et al. (2009), which is based on capacity scaling using column and row generation, remains competitive.

Finally, most papers that consider multi-period variants utilize heuristics, such as Papadimitriou and Fortz (2015), who propose a rolling horizon heuristic to solve practical problem instances. An exception is Petersen and Taylor (2001) who use dynamic programming to solve one specific instance, whose state space can be reduced significantly. We next provide a mathematical description of the problem we consider.

3. Problem Description and Formulation.

We consider a network of nodes $\mathcal{N}$ and arcs $\mathcal{A}$ and a set of commodities $\mathcal{K}$ that need to be routed from given origin nodes to given destination nodes during each time period $t \in T$. Each commodity should satisfy a period-dependent demand between its origin and its destination. Routing a commodity through an arc incurs a variable, commodity-specific cost. In addition, routing a commodity through an arc is possible only if this arc is opened in this period, or if it has been opened in an earlier period. Opening an arc in a specific period incurs a fixed cost. We introduce the following notation:

Parameters

f_{ij}^t, cost of opening arc (i,j) at the start of period t

c_{ij}^k, cost of sending one unit of commodity k through arc (i,j)

u_{ij}, capacity of arc (i,j)

d^{kt}, demand of commodity k in period t

O_k, D_k, origin and destination of commodity k, respectively

$b_i^k = 1,\ \text{if } i \equiv O_k; -1,\ \text{if } i \equiv D_k;\ 0,\ \text{otherwise}$

Decision Variables

x_{ij}^{kt}, fraction of d^{kt} that is directed through arc (i,j)

y_{ij}^t, $= 1$ if arc (i,j) is opened at the beginning of period t, 0 otherwise

The multi-period network expansion problem (M-NEP) is formulated as follows:

$$\min \sum_{t\in T}\sum_{k\in\mathcal{K}}\sum_{(i,j)\in\mathcal{A}} c_{ij}^k d^{kt} x_{ij}^{kt} + \sum_{t\in T}\sum_{(i,j)\in\mathcal{A}} f_{ij}^t y_{ij}^t \qquad [M-NEP] \tag{1}$$

$$\text{s.t} \sum_{j:(i,j)\in\mathcal{A}} x_{ij}^{kt} - \sum_{j:(j,i)\in\mathcal{A}} x_{ji}^{kt} = b_i^k, \qquad \forall i\in\mathcal{N}, \forall k\in\mathcal{K}, \forall t\in T, \tag{2}$$

$$\sum_{k\in\mathcal{K}} d^{kt} x_{ij}^{kt} \le u_{ij} \sum_{l=1}^{t} y_{ij}^l, \qquad \forall (i,j)\in\mathcal{A}, t\in T, \tag{3}$$

$$x_{ij}^{kt} \le \min\{1, \frac{u_{ij}}{d^{kt}}\} \sum_{l=1}^{t} y_{ij}^l, \qquad \forall (i,j)\in\mathcal{A}, \forall k\in\mathcal{K}, \forall t\in T, \tag{4}$$

$$\sum_{t\in T} y_{ij}^t \le 1, \qquad \forall (i,j)\in\mathcal{A}, \tag{5}$$

$$0 \le x_{ij}^{kt} \le 1, \qquad \forall (i,j)\in\mathcal{A}, k\in\mathcal{K}, t\in T, \tag{6}$$

$$y_{ij}^t \in \{0,1\}, \qquad \forall (i,j)\in\mathcal{A}, \forall t\in T. \tag{7}$$

The objective function (1) minimizes the costs of routing commodities and opening arcs throughout the horizon. Constraints (2) maintain the balance of each commodity in each node and period. Constraints (3) prevent the total amount of flow that is routed through each arc from exceeding that arc's capacity in each period. If at time t, $y_{ij}^s = 0$, for all $1 \le s \le t$, then no flow is routed through the arc (i,j) during period t. Constraints (4) are redundant but potentially useful, since they strengthen the problem's LP relaxation. Specifically, they are the multi-period counterparts of the "strong" inequalities used in single-period problems to improve the LP relaxation (Gendron and Crainic 1994). Finally, constraints (5) express that each arc can be opened at most once. Arc capacity expansions can be modeled by considering additional pairs of arcs between nodes. This is important for rail networks,

where constructing additional tracks to expand the capacity between stations is commonplace (Bärmann et al. 2017).

To avoid trivial solutions, we assume that opening an arc earlier implies a higher cost, i.e., $f_{ij}^t > f_{ij}^{t+1}, \forall (i,j) \in \mathcal{A}, t \in T$. A special case of the above formulation arises when $u_{ij} \geq \sum_{k \in \mathcal{K}} d^{kt}$ for all $(i,j) \in \mathcal{A}$ and $t \in T$. Then, constraints (3) are redundant, and (4) change to $x_{ij}^{kt} \leq \sum_{l=1}^{t} y_{ij}^l, \forall (i,j) \in \mathcal{A}, k \in \mathcal{K}$ and $t \in T$. We will hereafter refer to this case as the multi-period *uncapacitated* network expansion problem (M-UNEP). This variant is interesting in its own right, because it has different decomposability features than the capacitated variant. Before applying decompositions, we introduce a heuristic that generates high-quality solutions fast.

4. Initial Heuristic Search.

Our heuristic tries to detect arcs that are going to be opened in some period, and then decides when it is best to open each of the identified arcs. Algorithm 1 describes the high-level steps of this select-and-time procedure.

Selecting good arcs. We first solve two single-period instances. The first one outputs a good set of candidate arcs, by incorporating cost and demand information from the entire horizon, and the second one adopts a worst-case demand perspective, to make sure the proposed set of arcs leads to feasible solutions. We start by calculating weights w^t that are used to average the fixed cost of each arc over the horizon. The weight of each period is determined in two steps. First, we set it equal to the fraction of the remaining cumulative demand over the total demand, and then we normalize between zero and one (line 1). The first single-period model we solve has a fixed cost per arc (i,j) that is a weighted average of that arc's cost across the horizon, i.e., $\hat{f}_{ij} = \sum_t w^t f_{ij}^t$ (line 2). In order to capture future demand growth, we use the first period demand of each commodity scaled by $\max_t d^{kt} / \frac{1}{T} \sum_l d^{kl}$ (line 3). That way we obtain an instance with fixed costs that are more representative of the entire horizon, and demands that anticipate the growth of future periods.

After solving this problem, we store which arcs are opened (line 4). These arcs may not be enough to guarantee feasibility throughout the horizon, and to tackle this we create an instance that has no arc opening cost and maximum demand from each commodity (line 5). This can be seen as a "worst-case" scenario from a demand perspective, because all commodity demands take their maximum values simultaneously. This instance can be solved efficiently as follows: since there are no arc opening costs, we solve an LP that allows positive flows via all arcs, and then select those arcs that have a strictly positive flow in the optimal solution. After solving this formulation (line 6), we form the set of arcs that are selected in either the first ($\hat{y}_{ij}$) or the second ($\bar{y}_{ij}$) model (line 7). After this initial selection of arcs, i.e., $\mathcal{A}^{pot}$, is determined, the decision of *when* to open an arc is considered.

Arc opening timing. In order to decide when to open arcs, we iteratively solve single-period instances for each period in the horizon, using only arcs from the set $\mathcal{A}^{pot}$, whose fixed cost is a weighted average of each period's cost and the average cost of the remaining periods, and the variable cost is inflated by $\max_t d^{kt} / \frac{1}{|T|} \sum_t d^{kt}$ (lines 9 - 11). This way we take into account the variable cost of future periods, anticipating the potential increase in demand. Every time an arc is newly opened, we mark it as such and do not consider its cost further (lines 15 - 16). During early periods, we give more weight to the actual fixed costs rather than the anticipated future costs, acknowledging that early capacity opening decisions are more important, since they impact the routing decisions of the entire horizon. Thus, fixed costs for the first periods carry the largest weights, while for later periods the weights of each period become smaller. Having decided when to open each arc, the problem reduces to solving a series of linear multi-commodity flow problems for each period in the planning horizon (line 20).

The advantage of the select-and-time heuristic is that although it works with a reduced problem size, it takes into account information from multiple periods. We thus use it to efficiently warm-start the decomposition schemes we develop.

Algorithm 1 The Select-and-Time heuristic procedure

Input: $d^{kt}, f^t_{ij}, c^k_{ij}, u_{ij}, b^k_i$

Output: $y^t_{ij}, x^{kt}_{ij}, v_H$

$w^t \leftarrow \frac{\sum_{l=t}^{|T|}\sum_{k\in\mathcal{K}} d^{kl}}{\sum_{l\in T}\sum_{k\in\mathcal{K}} d^{kl}}; w^t \leftarrow \frac{w^t}{\sum_{l\in T} w^l}$

$\hat{f}_{ij} \leftarrow \sum_{t\in T} w^t f^t_{ij}$ ▷ Create weighted fixed cost ($\hat{f}_{ij}$)

$\hat{d}^k \leftarrow d^{k1}\frac{\max_{t\in T} d^{kt}}{\frac{1}{|T|}\sum_{t\in T} d^{kt}}$ ▷ Create inflated demand ($\hat{d}^k$)

$\hat{y}_{ij} \leftarrow \textsc{SlvSnPer}(\hat{d}^k, \hat{f}_{ij}, c^k_{ij}, u_{ij}, b^k_i)$ ▷ Solve MIP with $\hat{f}_{ij}$, $\hat{d}^k$; Store $\hat{y}_{ij}$

$\hat{d}^k \leftarrow \max_{t\in T} d^{kt}; \hat{f}_{ij} \leftarrow 0$ ▷ Take max demand ($\hat{d}^k$), set zero fixed costs ($\hat{f}_{ij}$)

$\bar{y}_{ij} \leftarrow \textsc{SlvSnPer}(\hat{d}^k, \hat{f}_{ij}, c^k_{ij}, u_{ij}, b^k_i)$ ▷ Solve LP with $\hat{f}_{ij}$ $\hat{d}^k$; Store arcs ($\bar{y}_{ij}$)

$\mathcal{A}^{pot} = \{(i,j)\in\mathcal{A} \mid \hat{y}_{ij} + \bar{y}_{ij} \geq 1\}; \mathcal{A}^0 = \mathcal{A}\backslash\mathcal{A}^{pot}; \mathcal{A}^1 \leftarrow \emptyset$

for $t \in T$ **do**

  $t^{max} \leftarrow \min\{t+1, |T|\}$

  $\hat{f}_{ij} \leftarrow \{w^t f^t_{ij} + \frac{1-w^t}{|T|-t^{max}+1}\sum_{l=t^{max}}^{|T|} f^l_{ij},\ \forall(i,j)\in\mathcal{A}^{pot};\ 0, \forall(i,j)\in\mathcal{A}^1\}$

  $\hat{c}^k_{ij} \leftarrow c^k_{ij}\frac{\max_t d^{kt}}{\frac{1}{|T|}\sum_t d^{kt}}; \hat{d}^k \leftarrow d^{kt}$

  $\hat{y}^t_{ij} \leftarrow \textsc{SlvSnPer}(\hat{d}^k, \hat{f}_{ij}, \hat{c}^k_{ij}, u_{ij}, b^k_i \mid y_{ij}=1, \forall(i,j)\in\mathcal{A}^1; y_{ij}=0, \forall(i,j)\in\mathcal{A}^0)$

  ▷ Fix opened & closed arcs; solve for period t; store $\hat{y}^t_{ij}$

  for $(i,j)\in\mathcal{A}^{pot}$ **do** ▷ Find open arcs

    if $\hat{y}^t_{ij} = 1$ **then**

      $\mathcal{A}^{pot} \leftarrow \mathcal{A}^{pot}\backslash\{(i,j)\}$ ▷ If opened, remove from potential

      $\mathcal{A}^1 \leftarrow \mathcal{A}^1 \cup \{(i,j)\}$ ▷ Save opening

    end if

  end for

end for

$(y^t_{ij}, x^{kt}_{ij}, v_H) \leftarrow \textsc{SlvMltPerLP}(d^{kt}, f^t_{ij}, c^k_{ij}, u_{ij}, b^k_i \mid y^t_{ij} = \hat{y}^t_{ij}, \forall(i,j)\in\mathcal{A}, \forall t\in T)$

5. Decomposition Algorithms and Local-Search Heuristics.

We next exploit structural characteristics of capacitated and uncapacitated variants to utilize Lagrange relaxation and Benders decomposition, respectively.

5.1. Capacitated Problems.

5.1.1. Lagrange relaxation of M-NEP. By dualizing constraints (2) in the objective function (1), M-NEP decomposes into a series of single-arc multi-period subproblems. We let π_i^{kt} denote the dual values associated with constraints (2). To miminize notation clutter, we remove the indices (i, j) and formulate the single-arc problems as follows:

$$v_{ij}^{\pi} = \min \sum_{t\in T}\sum_{k\in\mathcal{K}}(c^k d^{kt} + \pi_i^{kt} - \pi_j^{kt})x^{kt} + \sum_{t\in T} f^t y^t \qquad [SUB] \tag{8}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} d^{kt}x^{kt} \le u\sum_{l=1}^{t} y^l, \qquad \forall t \in T \tag{9}$$

$$x^{kt} \le \min\{1, \frac{u}{d^{kt}}\}\sum_{l=1}^{t} y^l, \qquad \forall k \in \mathcal{K}, \forall t \in T \tag{10}$$

$$\sum_{t\in T} y^t \le 1, \tag{11}$$

$$0 \le x^{kt} \le 1, \qquad \forall k \in \mathcal{K}, t \in T \tag{12}$$

$$y^t \in \{0, 1\}, \qquad \forall t \in T. \tag{13}$$

Then, the Lagrange dual optimization problem can be expressed as

$$v_{LR}^* = \max_{\pi} v(\pi) = \max_{\pi} \left\{ \sum_{(i,j)\in\mathcal{A}} v_{ij}^{\pi} - \sum_{t\in T}\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{K}} b_i^k \pi_i^{kt} \right\}. \tag{14}$$

It is well-known that (14) is a concave optimization problem and that $v(\pi)$ is piece-wise linear (Fisher 1981). In order to calculate v_{LR}^*, we evaluate $v(\pi)$ pointwise and apply subgradient optimization. To this end, we note the following remark.

REMARK 1. For a given vector with $y^t \in \{0,1\},\ t \in T$ values that satisfy (11), problem $[SUB]$ decomposes into a series of single-period, linear bounded knapsack problems, each of which can be solved in $\mathcal{O}(|\mathcal{K}|\log|\mathcal{K}|)$ time. Given such a vector, $[SUB]$ can be solved in $\mathcal{O}(|\mathcal{K}||T|\log|\mathcal{K}|)$ time.

We next provide information on how strong the bound obtained by the Lagrange relaxation is. To this end, we note that the linear relaxation of [SUB] has the *integrality property*, i.e., its basic feasible solutions have $y^t \in \{0,1\}$ for all $t \in T$ without imposing the integrality restrictions explicitly (Fisher 1981).

PROPOSITION 1. *Problem [SUB] has the integrality property, thus* $v^*_{LP} = v^*_{LR}$.

Proof. See Appendix A of the electronic companion.

5.1.2. Lower Bounds and Approximate Primal Solutions. When solving the Lagrange dual problem (14) with regular subgradient optimization, the obtained primal solution typically violates the dualized constraints (2). Moreover, devising good quality branching rules from such a solution is challenging, because the arc opening decisions obtained by subgradient optimization (y^t_{ij}) are integral. To tackle this issue, we employ the volume algorithm of Barahona and Anbil (2000), an extension of the subgradient algorithm that returns, alongside the Lagrangian lower bound, a y-solution with small violations of (2). This way, we have a solution with fractional binary variables at each subgradient iteration, which we then use to direct our heuristic search.

5.1.3. Finding Upper Bounds. Within the framework of the volume algorithm we find feasible solutions of the original problem, M-NEP, by employing three local search heuristics. The first heuristic checks if moving the arc opening decisions later in time provides an improved solution. Algorithm 2 shows the details. For large models, however, attempting to check all arcs might be prohibitive. We thus restrict the search to a neighborhood defined by comparing the arc-opening variables of

Algorithm 2 Single-arc search heuristic

Input: Period t, feasible solution $(\bar{y}_{ij}^t, \bar{x}_{ij}^{kt})$, objective value z^t

Output: Improved feasible solution $(\bar{y}_{ij}^t, \bar{x}_{ij}^{kt})$, objective value z^t

$\bar{\mathcal{A}} \leftarrow \{(i,j) \in \mathcal{A} : \bar{y}_{ij}^t = 1\}$
$M^t \leftarrow$ LP model for period t with $x_{ij}^{kt} \leq \sum_{l=1}^{t} \bar{y}_{ij}^l, \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K}$
for $(i,j) \in \bar{\mathcal{A}}$ **do**
    $\Delta fc \leftarrow f_{ij}^t - f_{ij}^{t+1}$
    Solve M^t with $x_{ij}^{kt} = 0, \forall k \in \mathcal{K}$
    $z' \leftarrow$ Objective(M^t) if M^t is feasible, otherwise $z' = \infty; \Delta z \leftarrow z' - z^t$
    If $\Delta fc - \Delta z > 0$ **then** ▷ Fixed cost reduction > flow cost increase
        $\bar{y}_{ij}^t \leftarrow 0, \bar{y}_{ij}^{t+1} \leftarrow 1, \bar{x}_{ij}^{kt} \leftarrow$ Solution$(M^t), z^t \leftarrow \sum_{(i,j) \in \bar{\mathcal{A}}} c_{ij}^k d^{kt} \bar{x}_{ij}^{kt}$
end for
return $(\bar{y}_{ij}^t, \bar{x}_{ij}^{kt}, z^t)$

the best feasible solution (the incumbent) and the possibly fractional arc-opening variables, retrieved by the volume algorithm. Concretely, for a cutoff point $\delta \in (0,1)$ and for each arc (i,j), we set $y_{ij}^{t^*} = 1$ if $t^* = \arg\max_t \{y_{ij}^t : y_{ij}^t > \delta\}$ exists and $y_{ij}^t = 0, \forall t \in T$, otherwise. Then, we compare the resulting vector with the incumbent solution. For each period, we invoke Algorithm 2 only for the arcs that are different in the incumbent and the rounded solution. In addition, we sort the arcs in order of decreasing Δfc, in order to ensure that we first check those that give the highest period-on-period reduction in fixed costs. This procedure utilizes a small set of relevant arcs, for which there are discrepancies between the rounded approximate primal solution of the Lagrange relaxation and the incumbent solution.

Our second heuristic checks if changing the period when an arc is opened leads to improved solutions. Thus, given a feasible solution $\bar{y}_{ij}^t$ we define $\bar{\mathcal{A}}_t = \{(i,j) \in$

$\mathcal{A}|\bar{y}^t_{ij} = 1\}, \forall t \in T$ and impose the constraints

$$\sum_{l=\max\{t-\tau^-,0\}}^{\min\{|T|,t+\tau^+\}} y^l_{ij} = 1, \forall t \in T, \forall (i,j) \in \bar{\mathcal{A}}_t \tag{15}$$

to the original problem M-NEP. This explores a neighborhood of the incumbent where the arc opening decisions remain the same, but their timing can be shifted up to τ^+ periods later or τ^- periods earlier.

Our third and last heuristic applies a simple fixing procedure based on the values of the fractional y variables recovered by the volume algorithm, as follows. First, it interprets each fractional value as signifying a degree of "ambiguity", meaning that the exploration of variables closer to 0.5 takes priority. To this end, it sorts the y variables in increasing values of $|y^t_{ij} - 0.5|$ and, for a given fraction f, it finds l_f and u_f in $(0, 1)$ such that the first $f|\mathcal{A}||T|$ variables lie in the interval $[l_f, u_f]$. Finally, in M-NEP, it fixes to zero those variables that are lower than l_f and to one those that are higher than u_f, respectively, and solves the remaining MIP model. The advantage of this heuristic is that it can be tuned easily by only changing f, while it searches a promising neighborhood of the fractional solution.

5.2. Uncapacitated Problems.

Removing constraints (3) from M-NEP gives rise to the uncapacitated variant, M-UNEP. The resulting model exhibits a decomposable structure: for fixed arc opening decisions, it decomposes into a series of independent shortest path problems, per commodity and per period. In this part, we leverage this property to develop Benders decomposition formulations.

5.2.1. Regular Benders Decomposition. Let Y denote the set of binary vectors y that satisfy $\sum_{t\in T} y^t_{ij} \leq 1, \forall (i,j) \in \mathcal{A}$. Then, for a given $\bar{y} \in Y$, the uncapacitated model reduces to the following linear program:

$$z(\bar{y}) = \min \sum_{t\in T} \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c^k_{ij} d^{kt} x^{kt}_{ij} \tag{16}$$

$$\text{s.t} \sum_{j:(i,j)\in\mathcal{A}} x_{ij}^{kt} - \sum_{j:(j,i)\in\mathcal{A}} x_{ji}^{kt} = b_i^k, \qquad [\pi_i^{kt}],\ \forall i \in \mathcal{N}, \forall k \in \mathcal{K}, \forall t \in T, \tag{17}$$

$$0 \leq x_{ij}^{kt} \leq \sum_{l=1}^{t} \bar{y}_{ij}^{l}, \qquad [\lambda_{ij}^{kt}],\ \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K}, \forall t \in T, \tag{18}$$

Then, the original problem can be expressed as $\min_{y\in Y}\{z(y)+\sum_{t\in T}\sum_{(i,j)\in\mathcal{A}} f_{ij}^t y_{ij}^t\}$. Note that (16)-(18) decomposes into a series of shortest path problems, each one corresponding to a commodity–period pair. The dual of (16)-(18) is then

$$z(\bar{y}) = \max \sum_{t\in T}\sum_{k\in\mathcal{K}} \left(\pi_O^{kt} - \pi_D^{kt}\right) - \sum_{(i,j)\in\mathcal{A}}\sum_{k\in\mathcal{K}}\sum_{t\in T}(\sum_{l=1}^{t} \bar{y}_{ij}^{l})\lambda_{ij}^{kt} \tag{19}$$

$$\text{s.t. } \pi_i^{kt} - \pi_j^{kt} - \lambda_{ij}^{kt} \leq c_{ij}^k d^{kt}, \qquad \forall (i,j) \in \mathcal{A}, k \in \mathcal{K}, t \in T \tag{20}$$

$$\lambda_{ij}^{kt} \geq 0, \qquad \forall (i,j) \in \mathcal{A}, k \in \mathcal{K}, t \in T. \tag{21}$$

The dual polyhedron $\Delta = \{(\boldsymbol{\pi}, \boldsymbol{\lambda}) | (20) - (21)\}$ is non-empty, and therefore can be represented by a finite set of extreme rays, denoted by R_Δ, and a finite set of extreme points, denoted by P_Δ (Schrijver 1998). Using this representation, the Benders reformulation of M-UNEP is as follows:

$$\min \sum_{t\in T}\sum_{(i,j)\in\mathcal{A}} f_{ij}^t y_{ij}^t + z \tag{22}$$

$$\text{s.t.} \sum_{t\in T}\sum_{k\in\mathcal{K}} \left(\bar{\pi}_O^{kt} - \bar{\pi}_D^{kt}\right) - \sum_{(i,j)\in\mathcal{A}}\sum_{k\in\mathcal{K}}\sum_{t\in T}\left(\sum_{l=t}^{|T|} \bar{\lambda}_{ij}^{kl}\right) y_{ij}^t \leq 0, \qquad \forall (\bar{\pi}_i^{kt}, \bar{\lambda}_{ij}^{kt}) \in R_\Delta \tag{23}$$

$$\sum_{t\in T}\sum_{k\in\mathcal{K}} \left(\bar{\pi}_O^{kt} - \bar{\pi}_D^{kt}\right) - \sum_{(i,j)\in\mathcal{A}}\sum_{k\in\mathcal{K}}\sum_{t\in T}\left(\sum_{l=t}^{|T|} \bar{\lambda}_{ij}^{kl}\right) y_{ij}^t \leq z \qquad \forall (\bar{\pi}_i^{kt}, \bar{\lambda}_{ij}^{kt}) \in P_\Delta \tag{24}$$

$$\sum_{t\in T} y_{ij}^t \leq 1, \qquad \forall (i,j) \in \mathcal{A} \tag{25}$$

$$y_{ij}^{kt} \in \{0,1\}, \qquad \forall (i,j) \in \mathcal{A}, t \in T. \tag{26}$$

Constraints (23), the *feasibility cuts*, prevent the objective function of the dual problem (19)-(21) from being unbounded, and therefore the corresponding primal problem (16)-(18) from becoming infeasible. Constraints (24) are the *optimality cuts*, which impose that z corresponds to the optimal objective value function of the dual subproblem (19)-(21). Although this formulation is useful conceptually, the often large cardinalities of P_Δ and R_Δ make the a priori inclusion of the corresponding constraints inefficient from a computational point of view. Benders himself noted that formulation (22)-(26) can be solved for a limited number of feasibility and optimality cuts, in which case it delivers a lower bound on the optimal objective function value, and new cuts can be added dynamically. Specifically, in each iteration the optimal y_{ij}^t values can be used to solve the pair of primal-dual subproblems (16)-(18) and (19)-(21), respectively. If the primal subproblem is infeasible, then the dual subproblem returns a feasibility cut, (23), whereas when the primal subproblem is feasible, the dual subproblem returns an optimality cut, (24). These cuts are added to the master problem and the algorithm proceeds to the next iteration. If no optimality or feasibility cut is violated, then the algorithm has converged to an optimal solution.

Modern implementations of Benders decomposition employ additional computational enhancements. First, the decomposition of the subproblem into a series of $|\mathcal{K}||T|$ subproblems allows the generation of individual cuts from each commodity–period pair. To this end, z is replaced by $\sum_{k\in\mathcal{K}}\sum_{t\in T} z^{kt}$ in (22), the summations over periods and commodities are dropped in (23) and (24), and the sets of extreme rays and points are defined over the polyhedra $\Delta^{kt} = \{(\boldsymbol{\pi},\boldsymbol{\lambda}) \mid \pi_i^{kt} - \pi_j^{kt} - \lambda_{ij}^{kt} \leq c_{ij}^k d^{kt}; \lambda_{ij}^{kt} \geq 0, \forall (i,j) \in \mathcal{A}\}$. The corresponding cuts are denser and tend to be more effective than a single cut generated from the aggregated master problem (22)-(26) (Cordeau et al. 2001). Second, instead of solving the master program (22)-(26) to optimality in each iteration, we take advantage of the callback capabilities of

modern solvers to solve it only once and add the cuts dynamically in the branch-and-bound tree. Concretely, we start by solving the master program with a limited number of cuts, use a callback to invoke the cut-generating procedure every time a feasible solution is found and to add the generated cuts, and then return control to the solver (Bai and Rubin 2009, Adulyasak et al. 2015). In addition, we add cuts from individual subproblems, and two classes of valid inequalities which warm-start the master problem, as detailed in Section 5.2.5.

5.2.2. FSZ Benders Decomposition. Fischetti et al. (2010) suggest an alternative normalization approach that uses the best known flow cost values $\bar{z}^{kt}$ in the subproblem formulation. Let $(\bar{y}^t_{ij}; \bar{z}^{kt})$ denote a feasible solution of the master problem (22)-(26) with disaggregated Benders cuts. For notational brevity, we suppress the commodity and period notation when we refer to a single subproblem. Using this notation, Fischetti et al.'s subproblem (FSZ) can be formulated as follows:

$$\max \quad \pi_O - \pi_D - \sum_{(i,j)\in\mathcal{A}} (\sum_{l=1}^{t} \bar{y}^l_{ij})\lambda_{ij} - \bar{z}\eta \qquad [FSZ] \tag{27}$$

$$\text{s.t.} \quad \pi_i - \pi_j - \lambda_{ij} \leq c_{ij} d\eta, \qquad \forall (i,j) \in \mathcal{A} \tag{28}$$

$$\sum_{(i,j)\in\mathcal{A}} w_{ij}\lambda_{ij} + w_0\eta = 1 \tag{29}$$

$$\eta \geq 0;\ \lambda_{ij} \geq 0, \qquad \forall (i,j) \in \mathcal{A}. \tag{30}$$

In this formulation, the user is able to select non-negative weights w_{ij} and w_0 to better configure the normalization hyperplane (29). The regular Benders subproblem arises as the special case of $w_0 = 1$ and $w_{ij} = 0$. Our implementation sets the convexity condition $w_0 = w_{ij} = 1$ for each arc $(i,j) \in \mathcal{A}$. As long as non-negative weights are selected, FSZ always has a feasible solution and is bounded. Therefore, a cut $\pi_O - \pi_D - \sum_{(i,j)\in\mathcal{A}} (\sum_{l=1}^{t} \bar{y}^l_{ij})\lambda_{ij} - \bar{z}\eta \leq 0$ can always be generated. Note also

that if no arc has been opened until period t, i.e., $\sum_{l=1}^{t} \bar{y}_{ij}^{l} = 0$ for each $(i,j) \in \mathcal{A}$, an optimal solution will generate a cut with $\eta = 0$ and $\sum_{(i,j)\in\mathcal{A}} \lambda_{ij} = 1$. Thus, this subproblem can generate both optimality and feasibility cuts.

5.2.3. Pareto-Optimal Cuts. The cut selection problem becomes relevant when the dual subproblem (19)-(21) has multiple optimal solutions, and therefore one has to select the best among alternative cuts. A criterion that partially quantifies cut quality is *cut dominance* (Magnanti and Wong 1981): a cut generated from the extreme point $(\boldsymbol{\pi}^1, \boldsymbol{\lambda}^1)$ is said to dominate another cut generated from $(\boldsymbol{\pi}^2, \boldsymbol{\lambda}^2)$ iff

$$\pi_O^1 - \pi_D^1 - \sum_{(i,j)\in\mathcal{A}} \left(\sum_{l=1}^{t} y_{ij}^{l} \right) \lambda_{ij}^1 \geq \pi_O^2 - \pi_D^2 - \sum_{(i,j)\in\mathcal{A}} \left(\sum_{l=1}^{t} y_{ij}^{l} \right) \lambda_{ij}^2$$

holds for all $y_{ij}^t \in Y = \{y_{ij}^t \in \{0,1\} \mid \sum_{t\in T} y_{ij}^t \leq 1, \forall (i,j) \in \mathcal{A}\}$ with strict inequality for at least one point. A cut is non-dominated, or *Pareto optimal* (PO) if there is no other cut that dominates it. Magnanti and Wong (1981) devised a mechanism that generates PO cuts by solving an additional linear program, formulated as follows. First, let $\boldsymbol{y}^r = (y_{ij}^{t,r})_{(i,j)\in\mathcal{A}}^{t\in T}$ denote a *core point*, i.e., a point that lies in the relative interior of $conv(Y)$. Then, denoting by $\bar{y}_{ij}^t$ the master problem solution and by $z^* = \max\{\pi_O - \pi_D - \sum_{(i,j\in\mathcal{A}} \sum_{l=1}^{t} \bar{y}_{ij}^l \lambda_{ij}^t : (\boldsymbol{\pi}, \boldsymbol{\lambda}) \in \Delta^{kt}\}$ the subproblem solution for the pair (k,t), one can identify a PO cut by solving the following subproblem:

$$\max \quad \pi_O - \pi_D - \sum_{(i,j)\in\mathcal{A}} \left(\sum_{l=1}^{t} y_{ij}^{l,r} \right) \lambda_{ij} \qquad [PO - SUB] \tag{31}$$

$$\text{s.t.} \quad \pi_i - \pi_j - \lambda_{ij} \leq c_{ij} d, \qquad \forall (i,j) \in \mathcal{A} \tag{32}$$

$$\pi_O - \pi_D - \sum_{(i,j)\in\mathcal{A}} \left(\sum_{l=1}^{t} \bar{y}_{ij}^{l} \right) \lambda_{ij} = z^* \tag{33}$$

$$\lambda_{ij} \geq 0 \qquad \forall (i,j) \in \mathcal{A}. \tag{34}$$

Constraint (33) ensures that the new point is an optimal solution to the original subproblem, while the objective function ensures that it is a PO cut.

Although finding a core point is $\mathcal{NP}$-hard in general (Papadakos 2008), the simple structure of Y makes it possible to characterize a family of core points: given a feasible point, $\bar{y}^t_{ij} \in Y$, the perturbed point $y^{t,r}_{ij} = \{1-\epsilon|T| \text{ if } \bar{y}^t_{ij} = 1; \epsilon, \text{ otherwise}\}$ is a core point for $\epsilon \in (0, 1/|T|)$. Selecting a small ϵ guarantees that we generate a core point which lies in the neighborhood of $\bar{y}^t_{ij}$. Thus, we make use of this selection policy in our implementation. We also note that an optimal solution to the regular Benders subproblem is feasible to the PO subproblem.

5.2.4. Efficient Generation of Pareto-Optimal Cuts. For single-period uncapacitated problems, Magnanti et al. (1986) have shown that a PO cut can be generated by solving a single minimum cost flow problem instead of two generally structured LPs. We show here that their main argument can be extended to our setting. To this end, we rewrite the dual of (31)-(34) as follows:

$$\max \quad d \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} - z^*x_0 \tag{35}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{A}_i^+} x_{ij} - \sum_{j\in\mathcal{A}_i^-} x_{ji} = b_i(1+x_0), \qquad \forall i \in \mathcal{N} \qquad [\pi_i] \tag{36}$$

$$0 \le x_{ij} \le x_0 \sum_{l=1}^{t} \bar{y}^l_{ij} + \sum_{l=1}^{t} y^{l,r}_{ij}, \qquad \forall (i,j) \in \mathcal{A} \ [\lambda_{ij} \ge 0], \tag{37}$$

where x_0 represents the dual price of constraint (33). Magnanti et al. (1986) observe that x_0 can be fixed to any value greater than or equal to $\sum_{(i,j)\in\mathcal{A}} \sum_{l=1}^{t} y^{l,r}_{ij}$ at an optimal solution. In our computational experiments, we set $x_0 = |\mathcal{A}|$. Therefore, the objective term z^*x_0 becomes a constant and (35)-(37) is recast as a minimum cost flow problem, while its dual solution corresponds to a PO cut.

5.2.5. Strengthening the Master Problem. We use two families of cutting planes that strengthen the formulation of the master problem. First, we employ origin-destination cuts, which impose that at least one arc from each origin and to each destination should be opened, respectively. To explain our second set of inequalities, let p^* denote the shortest path cost for a certain commodity-period pair and p^*_{ij} the shortest path cost when arc (i,j) is removed from the graph. Then, the cut $z \geq p^*_{ij} + (p^* - p^*_{ij}) \sum_{l=1}^{t} y^l_{ij}$ is a valid cut (Magnanti et al. 1986), since it expresses the fact that the flow cost for a commodity cannot be lower than the shortest path cost when the complete graph is considered ($\sum_{l=1}^{t} y^l_{ij} = 1$) or p^*_{ij} when arc (i,j) is not opened ($\sum_{l=1}^{t} y^l_{ij} = 0$). In our implementation, instead of adding $|\mathcal{T}||\mathcal{K}||\mathcal{A}|$ such inequalities, we find for each commodity the arc whose removal increases the shortest path cost the most and add inequalities only for this arc.

6. Computational Experiments.

We conduct experiments that assess the quality of (i) the lower bound obtained by Lagrange relaxation (LR); (ii) the upper bound returned by the Select and Time (S&T) heuristic; (iii) the LR algorithm integrated with S&T and the heuristics described in Section 5.1.3 and (iv) the various Benders implementations. In what follows, we report on the computational performance of our algorithms, broken down per number of nodes and periods for multiperiod instances constructed from the networks of Pazour et al. (2010), which originate from real transportation networks, and provide aggregate results for instances adapted from a subset of the artificial `R` instances. In total, our complete dataset consists of 1,080 instances. A detailed description of the data construction process and original instances, and the complete experiments on the `R` instances can be found in Appendices B and C of the online supplement of this paper, respectively. An additional analysis regarding the structural characteristics of solutions can be found at Fragkos et al. (2019).

All experiments were carried out using one thread on an Intel Xeon X5675 3.07GHz processor. All algorithms are coded in Python 2.7.10 using the NumPy library for numeric manipulations (van der Walt et al. 2011) and Gurobi v8.0.0 to solve the mixed-integer linear programs. We set a time limit of 7,200 seconds and 12GB of RAM for all methods. The source code of our implementations can be found at the public Github repository https://github.com/IoannisFragkos/Multi-period-network-design.

6.1. Instances.

Pazour instances. We employ the networks `USC30`*,* `USC53` *and* `JBH50` *from Pazour et al. (2010). The first two networks make use of the 2002 Commodity Flow Survey and the other network,* `JBH50`*, shows the 2002 annual shipments of J.B. Hunt transport services, a large US-based carrier. Table 1 shows the characteristics of these instances. We refer the interested reader to Pazour et al. (2010) for more information on the details of these networks.*

Instance	$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	$\|\mathcal{K}\|$
USC30	30	126	87
USC53	53	278	245
JBH50	50	198	62

Table 1 Characteristics of the networks of Pazour et al. (2010)

We use these networks to construct 648 instances with horizons varying from 5 to 20 periods, 624 of which are feasible for capacitated problems. Following Crainic et al. (2001), we construct instances with loose, medium and tight capacity and with low, medium and high fixed cost ratios. In addition, we further extend the design space by considering instances that have correlations between fixed and variable costs and instances where these costs can be proportional to the original distance matrix, random or mixed. More details regarding the construction details of these instances can be found in the online Appendix B.

R *instances.* The original R instances, which are randomly generated, are codified as rx.y, with x denoting the number of nodes, arcs and commodities, and y denoting the fixed cost and capacity combination that is considered. Table 2 shows this information for the instances we employed. More information about the single-period data generators can be found in Gendron and Crainic (1994, 1995).

x	Nodes	Arcs	Commodities
3	10	35	50
6	10	60	50
9	10	83	50
11	20	120	100
14	20	220	100
17	20	318	100

y	Cost	Capacity
1	L	L
2	M	L
3	H	L
4	L	M
5	M	M
6	H	M
7	L	T
8	M	T
9	H	T

Table 2 Taxonomy of the original R instances.

From these instances, we generate instances with 20, 40, 60 and 80 periods and with low and high demand variability. Under a full factorial design, 8 multi-period instances are generated from each single-period instance, accounting for 432 instances in total. From these instances, 408 are found feasible for capacitated problems, and all are feasible for the uncapacitated variant.

6.2. Capacitated Problem.

Lagrange dual: lower bound quality. In order to assess how close the LR bound is to the LP bound, we solve the LP relaxation of (1)-(7) with Gurobi, adding all strong inequalities as default constraints, and compare this bound with the one obtained by LR. Columns (1) to (6) of Table 3 report the results, showing in column (3) the number of instances for which Gurobi could solve the LP relaxation within a time limit of 7,200s, in column (4) the lower bound gap, defined as (LB_{GRB} −

$LB_{LR})/LB_{GRB}$, and the CPU time consumed by Gurobi and LR in columns (5) and (6), respectively, for the instances solved to optimality. These results suggest that the lower bound returned by LR is close to the exact bound, having a gap of approximately 1%. For the Pazour instances, Gurobi consumes 40% more CPU time, and although it is faster than LR for small instances, its performance does not scale up equally well for larger instances. To see this, note that for the 20-period instances Gurobi fails to solve the LP relaxation for 40 out of 134 instances, while for the remaining 94 instances it is marginally faster than LR. Furthermore, for the `R` instances with 20 to 80 periods, the CPU time of LR is one fifth of the CPU time for Gurobi.

Lagrange dual bound vs. Gurobi root node bound. Next, we compare the LR bound against the actual bound Gurobi returns at the root node, in columns (7) to (10) of Table 3. The reason for doing so is that Gurobi does not necessarily utilize all strong inequalities when solving the root node of the MIP formulation, but rather a subset of them combined with generic cutting planes. In fact, while the LP relaxation with strong inequalities could not be solved within the time limit for 130 Pazour instances and for 151 `R` instances, when solved as a MIP Gurobi was able to return a lower bound at the root node for all Pazour instances and 384 out of 408 `R` instances. However, since not all strong inequalities are included by default, this lower bound is no longer guaranteed to be better than the one of LR. This leads us to partition the instances in those where LR or GRB returned the best lower bound, respectively, and assess each category separately.

Column (7) reveals that in cases where LR returns a stronger lower bound for the Pazour instances, this is 8.42% stronger than that of Gurobi, while column 8 suggests that when Gurobi finds a stronger lower bound this is only 1.37% stronger than that of LR. These conclusions apply also to the `R` instances. Overall, Gurobi is four times slower for the Pazour and 20 times slower for the `R` instances, respectively, in calculating a root node lower bound compared to LR. These findings

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
Instances		GRB	LR<GRB LP	CPU Time (s)		LR>GRB Root	LR<GRB Root	CPU Time (s)	
$\|\mathcal{N}\|$	#	Solved	LB Gap (%)	GRB	LR	LB Gap (%)	LB Gap (%)	GRB	LR
30	216	216	0.43	**4**	147	0.09 (59)	0.61 (157)	**33**	147
50	164	97	0.86	1,301	**733**	16.11 (56)	1.17 (108)	3,652	**913**
53	216	181	2.16	991	**639**	9.68 (48)	2.20 (168)	3,473	**683**
$\|T\|$									
5	159	151	0.23	670	**278**	2.59 (29)	0.27 (130)	1,828	**289**
10	155	130	0.76	670	**422**	6.40 (52)	0.89 (103)	2,306	**478**
15	148	119	1.69	638	**578**	12.54 (41)	1.80 (107)	2,480	**681**
20	134	94	2.48	**447**	564	10.98 (41)	2.94 (93)	1,828	**808**
Total	596	494	1.15	620	**442**	8.42 (163)	1.37 (433)	2,276	**552**
Total (R)	408	257	0.85	760	**166**	12.83 (101)	1.10 (283)	4,258	**206**

Notes. Columns (4) and (8) are calculated as $(LB_{GRB} - LB_{LR})/LB_{GRB}$, while column (7) as $(LB_{LR} - LB_{GRB})/LB_{LR}$. Columns (9) and (10) report the average CPU time for instances in (7) and (8) combined. Numbers in parentheses denote the number of instances. For the Pazour instances, we report results for 596 out of 624 instances, because Gurobi ran out of memory or returned a segmentation fault at the remaining 28 instances.

Table 3 Average Lagrange relaxation gaps and CPU Time for instances where the exact lower bound is found (columns 1-6) and where Gurobi returns another lower bound at the root node (columns 7-10).

are qualitatively persistent across planning horizons of various lengths. Although Gurobi seems more efficient for the USC30 instances, which are the smallest ones in size and can all be solved optimally, its advantage is not retained for larger instances. The conclusion from this experiment is that LR delivers high quality lower bounds, while its performance scales well to large instances.

Upper Bounds. Next, we assess the quality of upper bounds, as obtained by the S&T heuristic and by the integrated LR algorithm, i.e., the LR bounding scheme combined with our incremental and local branching heuristics, when it is initialized by the S&T heuristic, as described in Section 5.1.3. To this end, we utilize the instances that Gurobi could solve to proven optimality, and compare their bounds to the ones obtained by our heuristics. In addition, we utilize two other benchmark

approaches: one that myopically solves single-period problems, fixes the arcs to be opened and proceeds to the next period, and a rolling horizon one which solves a problem of length $l < T$, fixes the arc opening decisions of the first m periods and repeats, starting from period $m+1$, until the end of the horizon. After some preliminary tuning, we select $l = 5$ and $m = 1$ when $T > 5$ and $l = 3, m = 1$ for $T = 5$. Table 4 reports the results.

		UB Gap (%)				CPU Time (s)				
$\|\mathcal{N}\|$	Optimal	MH	RH(5,1)	S&T	LR	MH	RH(5,1)	S&T	LR	GRB
30	216	14.37	1.30	4.09	**0.39**	12	107	**9**	189	128 (81)
53	117	12.98	2.75	4.80	**0.47**	767	2,077	**209**	1,277	1,389 (1,171)
50	89	6.59	3.78	2.99	**0.16**	854	1,749	**300**	1,599	1,415 (1,312)
$\|T\|$										
5	110	5.81	1.57	9.41	**0.35**	312	632	**92**	495	500 (439)
10	111	11.7	2.34	3.17	**0.36**	410	890	**142**	752	559 (598)
15	107	14.79	2.39	1.82	**0.36**	493	1,231	**162**	995	716 (849)
20	94	17.96	2.67	1.38	**0.41**	381	1,296	**103**	938	832 (699)
Total	422	12.34	2.23	4.06	**0.37**	399	999	**125**	788	649 (643)
Total (R)	163	27.88	2.85	4.84	**0.86**	249	1,634	**47**	332	2,092 (1,591)

Table 4 Upper bound quality of heuristics. Gaps are calculated as $(UB_{heur} - UB_{GRB})/UB_{GRB}$. The last column shows total time and, in parenthesis, the time Gurobi consumed to find the optimal solution.

A first observation is that the myopic heuristic (MH) delivers solutions that deviate considerably from optimality. In particular, solution quality deteriorates consistently for problems with longer horizons. The rolling horizon heuristic (RH) finds much better solutions, but also consumes a considerably larger CPU time, while its performance also deteriorates with longer horizons. The S&T heuristic finds solutions with a larger gap compared to RH, but it does so in a fraction of the time RH requires to terminate. Given that the purpose of S&T is to find good solutions to inject in LR, its performance is aligned with its objective. In addition, it delivers lower gaps for longer horizons and the main difference with RH is in

five-period instances. Finally, we note that LR finds solutions of excellent quality, having an average gap of 0.37% for the Pazour and 0.86% for the R instances, respectively. In terms of CPU time, LR is slightly slower than Gurobi for the small Pazour instances but six times faster for the R instances. The slower performance on the Pazour instances can be partly attributed to the fact that our algorithm is tuned to be efficient across a wider set of instances. Specifically, we made some critical design choices, such as the number of subgradient iterations, so that not only a good upper bound is found but also a strong lower bound is returned consistently across all instances, at the expense of longer CPU times for easier instances.

Finally, Table 5 reports the overall performance of the LR algorithm on all instances. Specifically, column 3 reports the number of instances for which Gurobi

	Instances		Integrality Gap (%)		CPU Time (s)		Only LR	
$\|\mathcal{N}\|$	All	GRB Gap<1	GRB	LR	GRB	LR	Gap (%)	CPU Time (s)
30	216	216	**0.00**	0.96	**128**	189	-	-
50	192	128	4.61	**2.11**	1,953	**1,726**	2.63	2,849
53	216	192	5.49	**4.26**	2,481	**1,608**	6.88	2,440
$\|T\|$								
5	159	153	3.13	**1.80**	2,121	**825**	7.48	2,033
10	156	141	3.18	**1.87**	1,966	**1,048**	2.70	2,838
15	155	129	3.03	**2.71**	1,836	**1,235**	4.09	2,949
20	154	113	**2.89**	3.58	1,854	**1,215**	4.75	2,532
Total	624	536	3.07	**2.41**	1,951	**1,065**	4.33	2,686
Total (R)	408	357	7.51	**5.14**	4,868	**517**	9.76	3,738

Table 5 Optimality gap quality of LR. Gaps are calculated as $(UB - LB)/UB$. The last two columns refer to instances for which Gurobi terminated after 7,200 seconds without an upper bound.

could find non-trivial upper and lower bounds, i.e., which have a gap less than 100%, and columns 4 and 5 the corresponding optimality gaps for each method, with respect to their own upper and lower bound. Next, columns 6 and 7 report the corresponding CPU times. The results suggest that LR delivers a better overall

gap, while it consumes less CPU time compared to Gurobi. This is true for all but the smaller instances, USC30, and for instances with 20 periods. For the latter, Gurobi returns a smaller gap, but note that it does not manage to return a gap for 41 instances. In both cases, LR remains competitive. It is worth noting that columns 3 to 7 subsume instances that Gurobi solved to optimality. If we consider instances which Gurobi could not solve to optimality but for which it did find an upper and lower bound, then the difference is even more profound: for example, for the Pazour instances LR returns an average gap of 5.09% and 2,088 seconds of CPU time, as opposed to a gap of 14.41% and 7,200 seconds of CPU, reported by Gurobi. Finally, the last two columns report the optimality gap and CPU time of LR when run on instances for which Gurobi could not find an upper bound. For these difficult instances, our algorithm seems to scale very well, with an average CPU time of 2,686 seconds (3,738 for the `R` instances) and an optimality gap of 4.33% (9.76% for the `R` instances). Interestingly, it seems that instances with five periods are quite challenging to solve.

In summary, these experiments suggest that the S&T heuristic and the Lagrange relaxation constitute efficient bounding techniques that are competitive with Gurobi for small instances, scale better for medium instances and attain good optimality gaps for the most difficult instances.

6.3. Uncapacitated Problem.

We investigate the efficiency of our Benders decomposition implementations by utilizing the same set of multi-period instances but without considering their capacity. Specifically, we benchmark (i) a basic implementation (B-Reg); (ii) Adding PO cuts (B-MW1); (iii) Adding PO cuts solving only one subproblem, as in Magnanti et al. (1986) (B-MW2) and finally (iv) the formulation of Fischetti et al. (2010) (B-F). All implementations exploit modern callback technology to avoid solving the master problem multiple times (Bai and Rubin 2009) and make use of the

cutting planes described in Section 5.2.5, while the Pareto-optimal formulations update the core point using Remark 3. In addition, all methods other than B-F use a subroutine that detects if a primal subproblem is infeasible, by either finding a path between each $o-d$ pair or by detecting an edge cut set, which is then used to generate a feasibility cut. This is not necessary for B-F, since the FSZ subproblem generates both optimality and feasibility cuts. Similar to the capacitated experiments, our basic benchmark is the Gurobi (GRB) formulation, which uses the cut pool to handle the separation of the strong inequalities (18). Table 6 shows the average optimality gaps and CPU times obtained by each method. The time limit is set to 7,200 seconds for all algorithms.

		Optimality Gap (%)					CPU Time (s)				
$\|\mathcal{N}\|$	#	GRB	B-REG	B-MW1	B-MW2	B-F	GRB	B-REG	B-MW1	B-MW2	B-F
30	216	**0.00**	0.03	0.01	0.48	**0.00**	**28**	622	777	86	**28**
50	216	48.92	23.40	9.12	3.30	**0.67**	3,264	5,161	5,175	4,171	**2,240**
53	216	17.5	11.57	5.92	3.57	**1.38**	3,852	6,650	6,818	4,809	**2,888**
$\|T\|$											
5	162	8.66	7.21	3.38	2.34	**1.21**	2,384	3,952	3,910	3,103	**1,959**
10	162	15.43	9.62	3.54	2.57	**0.68**	2,456	4,107	4,624	2,983	**1,710**
15	162	30.79	13.65	2.86	2.23	**0.49**	2,334	4,276	4,975	3,047	**1,465**
20	162	33.69	16.20	10.27	2.65	**0.35**	2,351	4,243	3,517	2,956	**1,630**
Total	648	22.14	11.67	5.01	2.45	**0.68**	2,381	4,144	4,257	3,022	**1,696**
Total (R)	432	15.12	6.47	2.33	1.52	**0.42**	1,902	2,191	2,866	1,583	**1,572**

Table 6 Optimality gaps and CPU times of Gurobi and Benders implementations.

A careful analysis of Table 6 suggests some important conclusions. First, Benders reformulations seem to be intrinsically more efficient compared to branch-and-cut (GRB). Specifically, our best implementation (B-F), attains an average gap which is approximately 30 times lower, and consumes about 70% of GRB's CPU time (80% for the R instances). This performance difference has been observed for other

problems as well, such as facility location (Fischetti et al. 2016), suggesting that a modern implementation of Benders decomposition can be a superior alternative to an off-the-shelf, state-of-the-art solver. Second, with respect to the basic Benders implementation, we note that although it achieves a far better gap than branch-and-cut, it falls behind the more sophisticated implementations by a large margin. Third, when assessing the impact of PO cuts, we observe a non-trivial gap improvement between the basic Benders B-R and the basic PO Benders B-MW1 implementations, implying that the impact of adding PO cuts significantly enhances performance. However, these improvements come at a high CPU time cost, because two LPs are solved to generate every optimality cut, resulting in B-MW1 having the worst time performance across all methods. Fourth, implementation B-MW2, which generates PO cuts using a single subproblem per iteration dominates B-MW1 in terms of both gap and CPU time performance. Finally, B-F is the best-performing implementation, having clearly the lowest gap and CPU time. In particular, it was able to solve optimally 99 (54 for R) instances that GRB failed to solve to optimality, and 58 (39 for R) instances that no other algorithm solved optimally. The ability of this model to incorporate the current flow cost in the cut-generating subproblem and to generate cuts that unify feasibility and optimality leads to important improvements over the Magnanti-Wong implementations. Finally, the breakdown per original network shows that the USC30 instances can all be solved to near-optimality rather easily, while JBH50 and USC53 are more challenging. In addition, problems with more periods are generally more challenging to solve, but interestingly B-F seems to attain a better gap performance for longer horizons, using about the same amount of CPU time.

On further analysis, it is interesting to investigate the number and type of cuts each Benders implementation generates. To this end, Figure 1 shows the average number of binding, optimality and feasibility cuts each algorithm generated in

a subset of 306 Pazour instances for which all algorithms could find an optimal solution. The average number of Benders cuts generated over a large number of

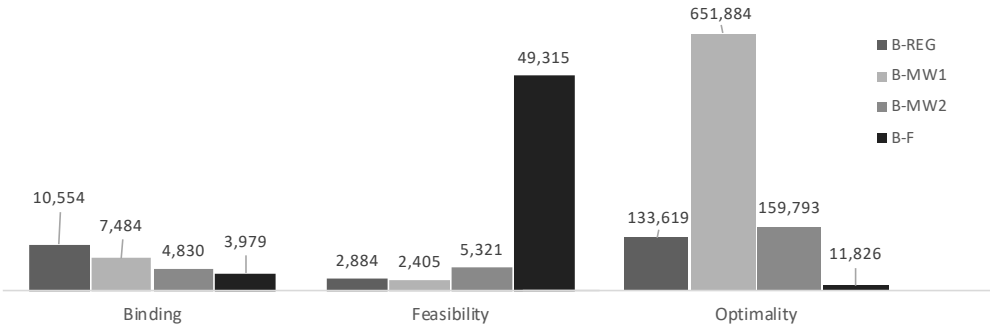


Figure 1 **Cuts generated by each implementation. Optimality and feasibility refer to the totals generated during branch-and-cut. Binding are the cuts binding at an optimal solution.**

instances may lead to some notable insights. First, in terms of cuts binding at an optimal solution, regular Benders is outperformed by all other algorithms, followed by B-MW1, B-MW2 and B-F. The regular implementation MW1, which solves two LPs to generate a single PO cut, generates a very large number of optimality cuts, and the smallest number of feasibility cuts. Interestingly, B-F generates the smallest number of binding and total optimality cuts, but it generates a very large number of feasibility cuts. A possible explanation for this is that it does not use the path finding subroutine that detects infeasibility, but rather generates feasibility cuts using the same subproblem as for optimality cuts. This experiment underlines that there may be a considerable spectrum when it comes to the performance variability of PO cuts. In particular, generating the cuts by solving LPs could result in strong cuts, since there exists some variability embedded in the pivoting rules that return one among multiple optimal solutions. An interesting direction for future research is to devise a mechanism that exploits this variability in a systematic way.

7. Extensions and Future Research.

We introduce a multi-period extension of the multi-commodity network design problem that arises in an array of applications, and study its capacitated

and uncapacitated variants. By taking advantage of the problem structure, we develop heuristic and decomposition-based algorithms that are superior to existing approaches. For capacitated problems, Lagrange relaxation scales very well with problem size and delivers high quality lower and upper bounds across a large range of problem sizes. For uncapacitated problems, we employ a variety of Benders decomposition formulations and show that the best one solves to optimality very large instances. Our algorithms are tested on two sets of networks.

A potential limitation of our study is that it assumes deterministic commodity demands. For freight rail applications, it is common to assume a target demand (Bärmann et al. 2017) which reflects anticipated growth and strategic targets. In other contexts, such as DHL's strategic service network design case (Cheung et al. 2001), deterministic network design may be combined with a simulation model to assess the network's operational characteristics and solution quality in a stochastic environment, although for such problems using a deterministic solution may lead to fewer consolidation opportunities (Lium et al. 2009). Further, deterministic solutions can be used to decide which arcs to open and when (Thapalia et al. 2012), and then solve the corresponding stochastic LPs. While conceptually simple, this heuristic may substantially improve the deterministic solution, by as much as 97% (Sun et al. 2017). For multi-period problems, where the routing costs are predominant over fixed costs, it is likely that such a method delivers good results.

There are several avenues for future research. For capacitated problems, volume-based branch-and-bound appears to be an approach worth exploring. Investigating the efficiency of well-established heuristics, such as large neighborhood search, evolutionary algorithms, slope scaling (Crainic et al. 2004) or column and row generation can lead to further improved solutions. For uncapacitated problems, further improvements on the Benders cut generation mechanism can be investigated. We hope that our study inspires future work on such problems.

Acknowledgments

This research was partly funded by the Fonds de recherche du Quèbec - Nature et technologies. This support is gratefully acknowledged. We also want to thank Dr. Jennifer Pazour, who provided us with the networks JBH50, USC30 and USC53.

References

Adulyasak Y, Cordeau JF, Jans R (2015) Benders decomposition for production routing under demand uncertainty. *Operations Research* 63(4):851–867.

Atamtürk A (2002) On capacitated network design cut–set polyhedra. *Mathematical Programming* 92(3):425–437.

Atamtürk A, Rajan D (2002) On splittable and unsplittable flow capacitated network design arc–set polyhedra. *Mathematical Programming* 92(2):315–333.

Bai L, Rubin P (2009) Combinatorial Benders cuts for the minimum tollbooth problem. *Operations Research* 57(6):1510–1522.

Balakrishnan A, Magnanti T, Wong R (1989) A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research* 37(5):716–740.

Barahona F, Anbil R (2000) The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming* 87(3):385–399.

Bärmann A, Martin A, Schülldorf H (2017) A decomposition method for multiperiod railway network expansion—with a case study for Germany. *Transportation Science* 51(4):1102–1121.

Baxter M, Elgindy T, Ernst AT, Kalinowski T, Savelsbergh MW (2014) Incremental network design with shortest paths. *European Journal of Operational Research* 238(3):675–684.

Bienstock D, Günlük O (1996) Capacitated network design-polyhedral structure and computation. *INFORMS Journal on Computing* 8(3):243–259.

Blanco V, Puerto J, Ramos A (2011) Expanding the Spanish high-speed railway network. *Omega* 39(2):138–150.

Cheung W, Leung L, Wong Y (2001) Strategic service network design for DHL Hong Kong. *Interfaces* 31(4):1–14.

Chouman M, Crainic T, Gendron B (2017) Commodity representations and cut-set-based inequalities for multicommodity capacitated fixed-charge network design. *Transportation Science* 51(2):650–667.

Cordeau JF, Soumis F, Desrosiers J (2001) Simultaneous assignment of locomotives and cars to passenger trains. *Operations Research* 49(4):531–548.

Costa A, Cordeau JF, Gendron B (2009) Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications* 42(3):371–392.

Crainic T, Frangioni A, Gendron B (2001) Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics* 112(1):73–99.

Crainic T, Gendreau M (2002) Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics* 8(6):601–627.

Crainic T, Gendreau M, Farvolden J (2000) A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing* 12(3):223–236.

Crainic T, Gendron B, Hernu G (2004) A slope scaling/Lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design. *Journal of Heuristics* 10(5):525–545.

Cruz F, Smith J, Mateus G (1998) Solving to optimality the uncapacitated fixed-charge network flow problem. *Computers & Operations Research* 25(1):67–81.

Fischetti M, Ljubić I, Sinnl M (2016) Redesigning Benders decomposition for large-scale facility location. *Management Science* 63(7):2146–2162.

Fischetti M, Salvagnin D, Zanette A (2010) A note on the selection of Benders' cuts. *Mathematical Programming* 124(1-2):175–182.

Fisher M (1981) The Lagrangian relaxation method for solving integer programming problems. *Management Science* 27(1):1–18.

Fragkos I, Cordeau JF, Jans R (2019) Decomposition methods for large-scale network expansion problems. CIRRELT Technical Report 2019-42, Université de Montréal, Canada.

Frangioni A, Gendron B (2009) 0–1 reformulations of the multicommodity capacitated network design problem. *Discrete Applied Mathematics* 157(6):1229–1241.

Frangioni A, Gendron B (2013) A stabilized structured Dantzig–Wolfe decomposition method. *Mathematical Programming* 140(1):45–76.

Frangioni A, Gorgone E (2014) Bundle methods for sum-functions with "easy" components: applications to multicommodity network design. *Mathematical Programming* 145(1-2):133–161.

Gendron B, Crainic T (1994) Relaxations for multicommodity capacitated network design problems. Technical report, Université de Montréal, cRT Technical Report 965, Université de Montréal, Canada.

Gendron B, Crainic T (1995) Bounding procedures for multicommodity capacitated network design problems. Technical report, Université de Montréal, cRT Technical Report 95-12, Université de Montréal, Canada.

Ghamlouche I, Crainic T, Gendreau M (2003) Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research* 51(4):655–667.

Günlük O (1999) A branch-and-cut algorithm for capacitated network design problems. *Mathematical Programming* 86(1):17–39.

Holmberg K, Yuan D (1998) A Lagrangean approach to network design problems. *International Transactions in Operational Research* 5(6):529–539.

Hooghiemstra J, Kroon L, Odijk M, Salomon M, Zwaneveld P (1999) Decision support systems support the search for win-win solutions in railway network design. *Interfaces* 29(2):15–32.

Kalinowski T, Matsypura D, Savelsbergh M (2015) Incremental network design with maximum flows. *European Journal of Operational Research* 242(1):51–62.

Katayama N, Chen M, Kubo M (2009) A capacity scaling heuristic for the multicommodity capacitated network design problem. *Journal of Computational and Applied Mathematics* 232(1):90–101.

Kim D, Pardalos P (1999) A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters* 24(4):195–203.

Lai Y, Shih M (2013) A stochastic multi-period investment selection model to optimize strategic railway capacity planning. *Journal of Advanced Transportation* 47(3):281–296.

Lee D, Dong M (2008) A heuristic approach to logistics network design for end-of-lease computer products recovery. *Transportation Research Part E: Logistics and Transportation Review* 44(3):455–474.

Lium A, Crainic T, Wallace S (2009) A study of demand stochasticity in service network design. *Transportation Science* 43(2):144–157.

Magnanti T, Wong R (1981) Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research* 29(3):464–484.

Magnanti TL, Mireault P, R T Wong RT (1986) Tailoring Benders decomposition for uncapacitated network design. *Netflow at Pisa*, 112–154 (Springer).

Marin A, Jaramillo P (2008) Urban rapid transit network capacity expansion. *European Journal of Operational Research* 191(1):45–60.

Minoux M (1989) Networks synthesis and optimum network design problems: Models, solution methods and applications. *Networks* 19(3):313–360.

Papadakos N (2008) Practical enhancements to the Magnanti–Wong method. *Operations Research Letters* 36(4):444–449.

Papadimitriou D, Fortz B (2015) A rolling horizon heuristic for the multiperiod network design and routing problem. *Networks* 66(4):364–379.

Paraskevopoulos D, Bektaş T, Crainic T, Potts C (2016) A cycle-based evolutionary algorithm for the fixed-charge capacitated multi-commodity network design problem. *European Journal of Operational Research* 253(2):265–279.

Pazour J, Meller R, Pohl L (2010) A model to design a national high-speed rail network for freight distribution. *Transportation Research Part A: Policy and Practice* 44(3):119–135.

Petersen E, Taylor A (2001) An investment planning model for a new north-central railway in Brazil. *Transportation Research Part A: Policy and Practice* 35(9):847–862.

Raack C, Koster A, Orlowski S, Wessäly R (2011) On cut-based inequalities for capacitated network design polyhedra. *Networks* 57(2):141–156.

Rahmaniani R, Crainic T, Gendreau M, Rei W (2016) The Benders decomposition algorithm: A literature review. *European Journal of Operational Research* 259(3):801–817.

Randazzo C, Luna H (2001) A comparison of optimal methods for local access uncapacitated network design. *Annals of Operations Research* 106(1-4):263–286.

Richards F (1959) A flexible growth function for empirical use. *Journal of Experimental Botany* 10(2):290–301.

Schrijver A (1998) *Theory of Linear and Integer Programming* (New York, NY: John Wiley & Sons), ISBN 0-471-90854-1.

Şeref O, Ahuja R, Orlin J (2009) Incremental network optimization: Theory and algorithms. *Operations Research* 57(3):586–594.

Sun C, Wallace S, Luo L (2017) Stochastic multi-commodity network design: The quality of deterministic solutions. *Operations Research Letters* 45(3):266–268.

Thapalia B, Wallace S, Kaut M, Crainic T (2012) Single source single-commodity stochastic network design. *Computational Management Science* 9(1):139–160.

van der Walt S, Colbert S, Varoquaux G (2011) The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering* 13(2):22–30.

Yaghini M, Karimi M, Rahbar M, Sharifitabar M (2014) A cutting-plane neighborhood structure for fixed-charge capacitated multicommodity network design problem. *INFORMS Journal on Computing* 27(1):48–58.

Yang H, Bell H, Michael G (1998) Models and algorithms for road network design: a review and some new developments. *Transport Reviews* 18(3):257–278.

Zhu E, Crainic T, Gendreau M (2014) Scheduled service network design for freight rail transportation. *Operations Research* 62(2):383–400.

Electronic Companion.

The electronic companion is organized in three appendices. The first appendix provides a proof and an illustration of Proposition 1. The second appendix provides details on the instances and the data construction process. Finally, the third appendix reports on the computational results of long-period instances.

Appendix A: Proof and Illustration of Proposition 1.

A.1. Proof.

We hereby prove that Problem [SUB] has the integrality property.

We will show that there exists no point with at least one fractional y^t that is an extreme point of [SUB]. To this end, note that the origin is a feasible point of [SUB]. Then, let a feasible point p be such that there exist s periods $\{t_1,\dots,t_s\}$ with $(y^{t_j})_p \in (0,1)$ and $(y^t)_p = 0$ for $t \notin \{t_1,\dots,t_s\}$. We construct feasible points $q_j, j=1,\dots,s$ such that the original point p can be written as their convex combination, using as weights the components of $(y^t)_p$, i.e., $(y^t)_p = \sum_{j=1}^{s}(y^{t_j})_p (y^t)_{q_j}$; $(x^{kt})_p = \sum_{j=1}^{s}(y^{t_j})_p (x^{kt})_{q_j}$ (1). To this end, for each $j \in \{1,\dots,s\}$ we set $(y^{t_j})_{q_j} = 1; (y^t)_{q_j} = 0$, for all $t \neq t_j$ and $(x^{kt})_{q_j} = \min\{1, A_j^{kt}\}$, if $t \geq t_j$; 0, otherwise, for all $k \in \mathcal{K}, t \in T$, where $A_1^{kt} = (x^{kt})_p/(y^{t_1})_p$ and $A_j^{kt} = \left[(x^{kt})_p - \sum_{l=1}^{j-1}(y^{t_l})_p (x^{kt})_{q_l}\right]/(y^{t_j})_p$ for all $j \in \{2,\dots,s\}$. Note that for each t and k there exists at least one j such that $A_j^{kt} \leq 1$, otherwise $(x^{kt})_{q_j} = 1$ for all j, and $A_s^{kt} > 1$ implies that $(x^{kt})_p > \sum_{j\in\{1,\dots s\}}(y^{t_j})_p$ which is not true because p is feasible. Thus, let j^* denote the first j such that $A_j^{kt} \leq 1$. We evaluate each $(x^{kt})_{q_j}$ for $j < j^*$, $j = j^*$ and $j > j^*$ and substitute them back in (1).
Case (i): $j < j^*$. By the definition of j^* and $(x^{kt})_{q_j}$ it holds that $(x^{kt})_{q_j} = 1$.
Case (ii): $j = j^*$. Likewise, the definitions of j^* and $(x^{kt})_{q_j}$ give $(x^{kt})_{q_{j^*}} = A_{j^*}^{kt}$ (2).
Case (iii): $j > j^*$. We first note that the quantities A_j^{kt} can be written recursively as $A_{j+1}^{kt} = \frac{(y^{t_j})_p}{(y^{t_{j+1}})_p}[A_j^{kt} - (x^{kt})_{q_j}]$ (3). This recursion written for $j = j^*$ implies that $A_{j^*+1}^{kt} = 0$ (because of (2)), which in turn gives $x_{q_{j^*+1}}^{kt} = 0$. Then, using (3) it is easy to show by induction that $A_j^{kt} = (x^{kt})_{q_j} = 0$ for all $j > j^*$.

Substitution of the above in (1) gives $\sum_{j=1}^{s}(y^{t_j})_p (x^{kt})_{q_j} = \sum_{j=1}^{j^*-1}(y^{t_j})_p + (y^{t_{j^*}})_p \underbrace{\left[(x^{kt})_p - \sum_{j=1}^{j^*-1}(y^{t_j})_p\right]/(y^{t_{j^*}})_p}_{A_{j^*}^{kt}} + \underbrace{0+\cdots+0}_{\text{Terms } \{j^*+1,\dots,s\}} = (x^{kt})_p$. This final relationship, alongside the fact that $\sum_j (y^{t_j})_p \leq 1$ and that the origin is feasible signifies that p can be written as a convex combination of q_i feasible points and the origin, which completes the proof.

We next provide a small example that shows how points q_j are constructed.

A.2. Illustration: Construction of Points q_j.

We illustrate hereby the construction of points q_j for a small example with one commodity and seven periods, in Table 7. There, it can be readily verified that $(y^t)_p = \sum_{j=1}^{3} (y)_{q^l} (y^t)_{q_l}$ and $(x^t)_p = \sum_{j=1}^{3} (y)_{q^l} (x^t)_{q^l}$, for all $t = 1, \ldots, 7$.

Point	Weight	Component	t=1	t=2	t=3	t=4	t=5	t=6	t=7
p	–	y		0.2		0.1			0.6
		x		0.2	0.1	0.25	0.25	0.05	0.8
q_1	$(y)_{q_1} = 0.2$	y		1					
		x		1	0.5	1	1	0.25	1
		A		1	0.5	1.25	1.25	0.25	4
q_2	$(y)_{q_2} = 0.1$	y				1			
		x				0.5	0.5		1
		A				0.5	0.5		6
q_3	$(y)_{q_3} = 0.6$	y							1
		x							0.83
		A							0.83

Table 7 Illustration of point construction. Empty spaces imply zero coordinates.

Appendix B: Data Construction Details.

For each instance that we generate in any of the datasets, we first check its feasibility. For capacitated problems, an instance is infeasible if there exists one period in which the corresponding single-period problem is infeasible. To check this, we run the myopic heuristic that solves single-period problems, specifically tuned to detect infeasibility. For uncapacitated instances, infeasibility detection reduces to checking whether there exists at least one path from each origin to each destination. We report details on how we constructed our instances next.

B.1. The Original Pazour Instances.

We employ the networks `USC30`, `USC53` and `JBH50` from Pazour et al. (2010). The first two networks make use of the 2002 Commodity Flow Survey, which is pursued jointly by the US Census Bureau, the US Department of Transportation, the Bureau of Transportation Statistics and the US Department of Commerce. For each origin-destination pair, we obtain the number of tons shipped

as commodity demand. The other network, `JBH50`, shows the 2002 annual shipments of J.B. Hunt transport services, a large US-based carrier. The unit of shipment used here is a 53-foot equivalent container. We refer the interested reader to Pazour et al. (2010) for more information on the details of these networks and focus hereby on our design decisions for the parts of the dataset that we generated.

We use these networks to construct 648 instances with horizons varying from 5 to 20 periods, 624 of which are feasible for capacitated problems. Pazour et al. (2010) use a single-period network design formulation but recognize that "*due to the high costs of these systems, it is likely that a high-speed network, [...], would be implemented in phases throughout a planning horizon of many years*". Therefore, these instances are appropriate use cases for our formulation.

In order to keep the instances tractable in a multi-period setting, we have kept commodities that cover 80% of the total original demand by eliminating the commodities with the smallest demand. Table 1 shows the characteristics of these instances. Gurobi is able to eliminate about 4% of variables via pre-processing, for capacitated instances using the M-NEP formulation. Therefore, the reported model sizes are accurate representations of the actual model sizes tackled by Gurobi. We used these three instances and the methodology in Crainic et al. (2001) to construct instances with loose, medium and tight capacities and with low, medium and high fixed cost ratios. In addition, we further extend the design space by considering instances that have correlations between fixed and variable costs and instances where these costs can be proportional to the original distance matrix, random or mixed. Table 8 presents in detail the levels of each parameter we considered. Using a full factorial design, we have constructed 54 instances from each original instance, which

Parameter	Levels	Symbols	Explanations
Fixed cost	Low, Medium, High	L, M, H	Ratio $fr = \frac{\sum f_{ij}}{\sum d^k \sum c_{ij}} \in \{0.01, 0.5, 0.1\}$, respectively
Capacity	Loose, Medium, Tight	L, M, T	Ratio $cr = \frac{\lvert\mathcal{A}\rvert \sum_{k\in\mathcal{K}} d^k}{\sum u_{ij}} \in \{1, 2, 8\}$, respectively
Correlation structure	Positive correlation (Original), Negative correlation	PC, NC	If negative, fixed and variable costs have correlation -70%
Routing cost	Euclidean, Mixed, Random	E, M, R	Costs proportional to distance (E), 50% of costs shuffled randomly (M), all costs shuffled randomly (R)

Table 8 Design parameters of single-period network design instances.

we then extended to multiple periods. The fixed cost per arc is assumed to decrease linearly with

the remaining periods, such that (i) the average fixed cost per arc equals that of the single-period instance, and (ii) the last period has 10% of the single-period fixed cost. For demand expansion, we use a sigmoid curve which consists of a convex, a linear and a concave part. This generic profile represents a period of rapid growth in demand, a subsequent linear trend and then a stabilization phase. The curves are constructed so that the average demand coincides with that of the original instance, and that demand expands from 50% to 150% of the original demand.

Variable Costs. For instances with Euclidean costs, we assume that variable routing costs are proportional to the physical distance of each arc. To construct instances with random costs, we shuffle randomly the original variable costs across the arcs. For instances with mixed costs, we randomly select 50% of arcs and reshuffle their variable costs.

Fixed Costs and capacities. To generate a wide range of fixed costs and capacities, we follow Crainic et al. (2001). First, we denote $T = \sum_{k \in \mathcal{K}} d^k$ and define the *capacity ratio* as $C = |\mathcal{A}|T / \sum_{(i,j) \in \mathcal{A}} u_{ij}$ and the *Fixed cost ratio* as $F = \sum_{(i,j) \in \mathcal{A}} f_{ij} / (T \sum_{(i,j) \in \mathcal{A}} c_{ij})$. We then generate nine instances from each original instance by combining $F \in \{0.01, 0.05, 0.1\}$ with $C \in \{1, 2, 8\}$ corresponding to low, medium and high fixed cost and loose, medium and tight capacity configurations, respectively. Each individual arc capacity was drawn from the discrete uniform distribution $\mathcal{U}\{0.5T/C, 1.5T/C\}$ and each individual fixed cost from the uniform discrete distribution $\mathcal{U}\{0.5TFc_{ij}, 1.5Fc_{ij}\}$. Thus, for instances with Euclidean costs, fixed costs are proportional to (and positively correlated with) variable costs. For instances with random costs we shuffle randomly the generated fixed costs across the arcs, while for mixed costs we randomly select 50% of arcs and reshuffle their fixed costs.

Correlations structure. In addition to the reshuffling operations, which influence the correlation between fixed and variable costs, we explicitly impose a condition in which fixed and variable costs have a strong negative correlation. To achieve this, we first check the resulting correlation between fixed and variable costs, since random shuffling could have resulted in a strong negative correlation, in which case we take no action. If the correlation is higher than -0.50, we rearrange the fixed costs so that the $n-th$ highest one is allocated to the arc with the $n-th$ lowest variable cost. As this creates a very strong negative correlation (close to -1), we then reshuffle a subset of fixed costs. Overall, the correlation coefficient of instances labeled 'NC' (Negative Correlation) is -0.70. Note that the difference with the cost mode, which can be Euclidian, Mixed and Random, is that imposing a cost mode means altering both fixed and variable costs, while imposing negative correlations is done by rearranging fixed costs only. For example, a random cost mode with a negative correlation means that (i) the originally generated fixed and variable costs have been randomly shuffled; (ii) if their

correlation was greater than -0.50 the fixed costs were remapped to arcs with the opposite order and (iii) a subset of the fixed costs has been reshuffled again, to mediate the negative correlation that has been created.

Extension to multiple periods. We construct multi-period fixed costs and demands based on their single period variants. Specifically, for fixed costs we consider a linear form in which the average cost per period equals the original value and the final period cost is 10% of the original fixed cost. The corresponding formula is $f_{ij}^t = [1.9 - 1.8(t+1)/T] f_{ij}$. For the demands, we utilize a generalized logistic function, which signifies a simple model of demand growth (Richards 1959), with additional random perturbation. Specifically, we use

$$d^{kt} = (1 + r^{kt}) \left(\lambda + \frac{\mu - \lambda}{(1 + e^{-B(t - \frac{T}{2})})^{\frac{1}{\nu}}} \right) d^{k1}, \forall k \in \mathcal{K}, 1 < t \leq T,$$

where r^{kt} is randomly drawn from $\mathcal{U}[-0.1, 0.1]$, λ and μ represent the minimum and maximum asymptotic demand respectively, B denotes the growth rate, which we set to 1, and ν is a location parameter, also set to 1, that determines near which asymptote the maximum growth occurs.

B.2. `R` Instances.

We have also created multi-period extensions of a subset of the Canad R dataset, which has been used extensively by several authors such as Crainic et al. (2000, 2001), Crainic and Gendreau (2002), Ghamlouche et al. (2003), Crainic et al. (2004), Costa et al. (2009). It is also used in Katayama et al. (2009) and Yaghini et al. (2014), which, to the best of our knowledge, seem to be the best-performing algorithms at the time of this writing.

The original `R` instances, which are randomly generated, are codified as `rx.y`, with `x` denoting the number of nodes, arcs and commodities, and `y` denoting the fixed cost and capacity combination that is considered. Table 9 shows this information for the instances we employed. More information about the single-period data generators can be found in Gendron and Crainic (1994, 1995).

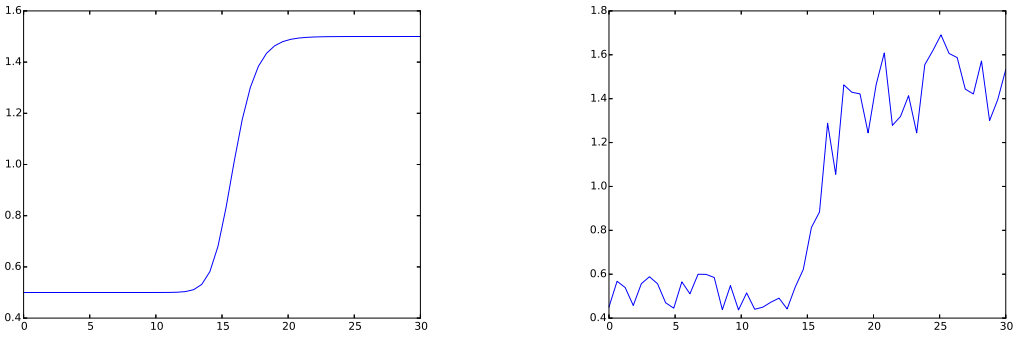
Extension to multiple periods. Extending the `R` instances over multiple periods was made using the same file generator, with one additional design level, namely demand variability. To this end, we constructed instances of small and large variability, with parameter r taking values in $\mathcal{U}[-0.1, 0.1]$ and $\mathcal{U}[-0.5, 0.5]$, respectively. Figure 2 shows an example of a logistic growth function with no perturbation to the left, and one with a high level of perturbation to the right.

From each original single-period instance we generate multi-period instances by varying (i) the number of periods and (ii) the demand variability of each instance. In particular, we generate instances with 20, 40, 60 and 80 periods and with low and high demand variability.

x	Nodes	Arcs	Commodities
3	10	35	50
6	10	60	50
9	10	83	50
11	20	120	100
14	20	220	100
17	20	318	100

y	Cost	Capacity
1	L	L
2	M	L
3	H	L
4	L	M
5	M	M
6	H	M
7	L	T
8	M	T
9	H	T

Table 9 Taxonomy of the original R instances.



(a) Logistic growth function for $\lambda = 0.5, \mu = 1.5$. (b) Logistic growth with random perturbations.

Figure 2 Demand is modeled using logistic growth functions and varying levels of perturbations.

Under a full factorial design, 8 multi-period instances are generated from each single-period instance, accounting for 432 instances in total. From these instances, 408 are found feasible for capacitated problems, and all are feasible for the uncapacitated variant. Table 10 gives an overview of their characteristics. Gurobi is able to eliminate about 1% of variables via pre-processing.

It is worth noticing that the R instances span across a wide variety of periods, nodes, arcs and commodities, resulting in a number of variables between 35,700 and 2,569,440 and a number of constraints between 10,720 and 185,520. Also important is that for capacitated instances the average optimality gap, i.e., the gap between the lower and upper bound, for the entire dataset is at 19%, between a minimum of 0% and a maximum of 100%, calculated by using the best known upper and lower bounds obtained by Gurobi for each instance. This comes as no surprise since even

# Instances	$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	$\|\mathcal{K}\|$	$\|T\|$	Bin Vars	Vars	Constraints
12	10	35	50	20	700	35,700	10,720
18	10	60	50	20	1,200	61,200	11,220
18	10	83	50	20	1,660	84,660	11,680
18	20	120	100	20	2,400	242,400	42,420
18	20	220	100	20	4,400	444,400	44,420
18	20	318	100	20	6,360	642,360	46,380
12	10	35	50	40	1,400	71,400	21,440
18	10	60	50	40	2,400	12,2400	22,440
18	10	83	50	40	3,320	169,320	23,360
18	20	120	100	40	4,800	484,800	84,840
18	20	220	100	40	8,800	888,800	88,840
18	20	318	100	40	12,720	1,284,720	92,760
12	10	35	50	60	2,100	107,100	32,160
18	10	60	50	60	3,600	183,600	33,660
18	10	83	50	60	4,980	253,980	35,040
18	20	120	100	60	7,200	727,200	127,260
18	20	220	100	60	13,200	1,333,200	133,260
18	20	318	100	60	19,080	1,927,080	139,140
12	10	35	50	80	2,800	142,800	42,880
18	10	60	50	80	4,800	244,800	44,880
18	10	83	50	80	6,640	338,640	46,720
18	20	120	100	80	9,600	969,600	169,680
18	20	220	100	80	17,600	1,777,600	177,680
18	20	318	100	80	25,440	2,569,440	185,520

Table 10 Instance characteristics. Constraint count excludes the strong inequalities. The number of instances per category may vary, since some instances were infeasible.

single-period capacitated problems are hard to solve to optimality, and most authors have designed heuristic approaches to tackle them.

Appendix C: Computational Experiments with Long-horizon Instances.

When it comes to the inclusion of the strong inequalities (4) in the LP relaxation of (1)-(7), we implemented and tested four alternative strategies: (i) not including strong inequalities; (ii) adding all of them a priori in the model; (iii) adding them dynamically (via a callback) when they are violated, and (iv) adding them in a lazy cut pool and resorting to Gurobi's pool management mechanism. Strategy (iv) gave the best results overall, when considering CPU time and optimality gap quality, and therefore we adopt this strategy throughout our computational experiments.

C.1. Capacitated Instances.

We conducted the same computational study described in the paper for the `R` instances. Specifically, for capacitated instances, we first assess the Lagrange relaxation lower bound quality, and then the quality of the upper bound obtained by our heuristics. Then, we proceed to further computational experiments, demonstrating the usefulness of our approach. Unless stated otherwise, the Lagrange relaxation algorithm performed 1,000 subgradient iterations and it was initialized using the select-and-time heuristic.

C.1.1. Lower Bound Quality. We first solved the LP relaxation of the problem with all the strong inequalities included. We then excluded 151 out of 408 instances in which Gurobi failed to solve the LP relaxation with the strong inequalities to optimality within the time limit of 7,200 seconds, since in those instances it did not return a lower bound. To this end, columns (3)-(6) of Table 11 report the lower bound gap, defined as $(LB_{GRB} - LB_{LR})/LB_{GRB}$ and CPU time obtained by Lagrange relaxation for the instances that Gurobi could calculate an exact lower bound.

We first observe that, overall, the lower bound obtained by Lagrange relaxation (LR) deviates less than 1% from the exact lower bound, obtained by including all inequalities (4), as the 0.85% gap indicates. When broken down by each input size, no large variability is apparent, while the gaps appear to be relatively insensitive to problem size. With respect to CPU time performance, LR is more than four times faster than Gurobi (GRB) overall, but the difference can be as large as 10 times.

In a second experiment, we defined the problem formulation as a MIP, added the strong inequalities as lazy constraints and let Gurobi solve the root node, having all standard cuts and the strong inequalities at its disposal. The idea here is that Gurobi may make use of some strong inequalities, and blend them with standard cutting planes and other techniques, such as reduced cost fixing, in order to improve the lower bound. As a result, the resulting lower bound may be lower or higher than the one calculated by the Lagrange relaxation. To this end, columns (7) and (8) of Table 11

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
# Instances		GRB	LR<GRB LP	CPU Time (s)		LR>GRB Root	LR<GRB Root	CPU Time (s)	
$\|\mathcal{N}\|$	All	Solved	LB Gap(%)	GRB	LR	LB Gap(%)	LB Gap(%)	GRB	LR
10	192	191	0.85	313	**124**	1.30 (1)	1.05 (191)	3,457	**124**
20	216	66	0.85	2,052	**288**	12.95 (100)	1.22 (92)	5,921	**376**
$\|\mathcal{A}\|$									
35	48	48	0.88	**64**	88	– (0)	0.94 (48)	320	**88**
60	72	72	0.66	326	**124**	– (0)	0.91 (72)	4,175	**124**
83	72	71	1.02	468	**148**	1.3 (1)	1.27 (71)	4,850	**148**
120	72	29	1.47	2,255	**213**	9.49 (26)	1.96 (46)	6,757	**271**
220	72	22	0.39	1,863	**335**	12.37 (35)	0.56 (27)	5,191	**423**
318	72	15	0.33	1,937	**364**	15.77 (39)	0.35 (19)	4,937	**566**
$\|\mathcal{K}\|$									
50	192	191	0.85	313	**124**	1.30 (1)	1.05 (191)	3,437	**124**
100	216	66	0.85	2,052	**288**	12.95 (100)	1.22 (92)	5,921	**376**
$\|T\|$									
20	102	87	0.83	935	**124**	5.56 (16)	1.05 (86)	3,591	**124**
40	102	65	0.9	606	**171**	12.89 (32)	1.20 (70)	4,240	**180**
60	102	56	0.83	647	**192**	15.27 (30)	1.13 (66)	4,577	**249**
80	102	49	0.83	781	**204**	14.63 (23)	1.05 (61)	4,875	**305**
Total	408	257	0.85	760	**166**	12.83	1.10	4,258	**206**

Notes. Columns (4) and (8) are calculated as $(LB_{GRB} - LB_{LR})/LB_{GRB}$, while column (7) as $(LB_{LR} - LB_{GRB})/LB_{LR}$. Columns (9) and (10) report the average CPU time for instances in (7) and (8) combined. Numbers in parentheses denote the number of instances.

Table 11 Average Lagrange relaxation gaps and CPU Time for instances where the exact lower bound is found (columns 1-6) and where Gurobi returns a positive lower bound at the root node (columns 7-10).

show the gaps between the LR lower bound and the GRB lower bound, when the former is stronger and when the latter is stronger, respectively, while columns (9) and (10) show the CPU time each method utilized. We observe that the LR lower bound remains consistently close to GRB's lower bound when GRB has a better bound, while when LR delivers a better lower bound GRB deviates 12.83%. In addition, LR is about 20 times faster, while it can be as much as 33 times faster.

To summarize, we notice that LR delivers high-quality lower bounds in cases where the optimal lower bounds can be verified, and very competitive lower bounds compared to Gurobi's default cutting plane algorithm. In both cases, the CPU time consumed by Lagrange relaxation is significantly lower, suggesting that it exhibits better scalability. Having established that our approach produces near-optimal lower bounds, we proceed to assess the quality of the upper bounds obtained by our heuristics.

C.1.2. Upper Bound Quality. In order to assess the upper bound quality obtained by our heuristics, we use formulation (1)-(6), but this time we let Gurobi solve it to optimality by using its default cutting planes, also including the strong inequalities in the lazy cut pool. Gurobi was able to solve 163 out of 408 instances to proven optimality within the time limit of 7,200 seconds, and we used the optimal solutions of these instances to assess the upper bounds obtained heuristically by four approaches: first, a myopic heuristic (MH) that solves each period independently, records which arcs are opened and uses them in the next periods and iterates; second, the rolling horizon heuristic that solves five-period problems (RH5), fixes the first period decisions and iterates; third, our select and time heuristic (S&T); and fourth, the hybrid bound obtained by S&T and Lagrange relaxation (S&T+LR). Table 12 reports how many instances were solved to optimality, the CPU times and upper bound gaps of each method, obtained as $(UB_m - OPT)/OPT$, where UB_m denotes the best upper bound found by each method $m \in \{MH, RH5, S\&T, S\&T + LR\}$.

The experiments show that the myopic heuristic is dominated by S&T, since it delivers solutions of worse quality and requires a higher amount of CPU time. On average, MH has an upper bound gap of 27.88%, and, while it requires a rather small amount of CPU time for small instances, it requires considerably high CPU times for instances with a large number of arcs, nodes or commodities. The S&T heuristic achieves gaps that are five times smaller, on average, while it needs a fifth of the time MH requires. The RH5 heuristic finds higher quality solutions than S&T, but in doing so consumes 35 times as much CPU time. Combined with Lagrange relaxation, S&T+LR achieves gaps lower than 1% on average, but this comes at the expense of a higher CPU time compared to S&T. However, S&T+LR is still about five times faster than RH5. In addition, S&T+LR is about six times faster than solving the problem to optimality, and about five times faster than finding the best solution using Gurobi. In summary, the S&T heuristic appears to strike a good balance between solution quality and CPU time, given that its goal is to initialize the Lagrange relaxation algorithm, while the hybrid S&T+LR scheme delivers high quality upper bounds in a time-efficient manner. We next turn our attention to a larger set of instances, namely those where Gurobi was able to calculate both a lower and an upper bound, but not necessarily an optimal solution.

	# Instances		Gap (%)				CPU Time (s)				
$\|\mathcal{N}\|$	All	GRB Opt	MH	RH5	S&T	S&T+LR	MH	RH5	S&T	S&T+LR	GRB
10	192	123	27.06	2.73	4.62	**0.71**	22	716	**14**	180	1,388 (998)
20	216	40	30.4	3.23	5.5	**1.32**	945	4,457	**150**	803	4,258 (3,412)
$\|\mathcal{A}\|$											
35	48	48	25.17	2.60	3.05	**1.12**	**4**	56	5	123	320 (292)
60	72	40	23.29	2.21	6.77	**0.45**	20	966	**13**	194	1,755 (1,195)
83	72	35	33.96	3.49	4.32	**0.45**	50	1,334	**27**	241	2,433 (1,742)
120	72	15	14.48	**1.50**	4.15	2.82	443	4,789	**93**	602	5,838 (4,525)
220	72	13	40.13	3.42	5.73	**0.56**	1,188	4,178	**195**	840	3,026 (2,513)
318	72	12	39.77	5.17	6.92	**0.28**	1,308	4,346	**174**	1,012	3,617 (2,996)
$\|\mathcal{K}\|$											
50	192	123	27.06	2.73	4.62	**0.71**	22	716	**14**	180	1,388 (998)
100	216	40	30.4	3.23	5.5	**1.32**	945	4,457	**150**	803	4,258 (3,412)
$\|T\|$											
20	102	51	20.26	0.91	2.23	**0.38**	198	1,240	**30**	177	1,113 (749)
40	102	42	26.32	2.49	4.44	**0.51**	300	1,890	**54**	329	2,267 (1,786)
60	102	37	32.86	4.14	6.32	**1.12**	267	1,775	**56**	409	2,522 (1,857)
80	102	33	36.07	4.86	7.71	**1.76**	240	1,757	**57**	492	2,901 (2,345)
Total	408	163	27.88	2.85	4.84	**0.86**	249	1,634	**47**	332	2,092 (1,591)

Notes. Gaps are calculated with respect to the optimal solution, as obtained by Gurobi. The time Gurobi needs to find the best solution is shown in parenthesis.

Table 12 Average upper bound gaps and CPU Times for instances solved to optimality.

C.1.3. Comparison with Branch-and-Cut. Although 163 instances were solved to proven optimality, a non-trivial optimality gap, i.e., a gap less than 100%, was calculated by Gurobi for a total of 357 instances. Therefore, 163 instances were solved to optimality and the remaining 357-163 = 194 were not solved to optimality but had a gap of less than 100%. We focus on these 357 instances to compare the efficiency of the hybrid S&T+LR algorithm with the branch-and-cut implementation of Gurobi. Correspondingly, columns (3)-(6) of Table 13 report the obtained results, where the gaps are calculated for each method using the corresponding upper and lower bounds.

	Instances		Integrality Gap (%)		CPU Time (s)		Only LR	
$\|\mathcal{N}\|$	All	GRB Gap<1	GRB	LR	GRB	LR	Gap (%)	CPU Time (s)
10	192	192	**1.05**	3.28	3,477	**209**	–	–
20	216	165	15.03	**7.3**	6,487	**874**	9.76	3,738
$\|\mathcal{A}\|$								
35	48	48	**0.00**	2.12	320	**115**	–	–
60	72	72	**1.76**	3.55	4,175	**217**	–	–
83	72	72	**1.03**	3.79	4883	**265**	–	–
120	72	56	6.33	**5.93**	6,836	**594**	5.37	2,661
220	72	55	15.64	**7.89**	6,214	**948**	13.05	3,627
318	72	54	23.43	**8.12**	6,404	**1,088**	10.54	4,801
$\|\mathcal{K}\|$								
50	192	192	**1.05**	3.28	3477	**209**	–	–
100	216	165	15.03	**7.3**	6,487	**874**	9.76	3,738
T								
20	102	102	4.49	**3.77**	4,157	**375**	–	–
40	102	101	10.14	**5.55**	5,149	**560**	4.74	572
60	102	85	9.01	**6.15**	5164	**572**	9.9	3,878
80	102	69	6.27	**5.31**	5145	**594**	9.83	3,762
Total	408	357	7.51	**5.14**	4,868	**517**	9.76	3,738

Table 13 Optimality gap quality of LR. Gaps are calculated as $(UB - LB)/UB$. The last two columns refer to instances in which Gurobi terminated after 7,200 seconds without an upper or lower bound (i.e., 100% gap).

Overall, Lagrange relaxation achieves a lower gap than branch-and-cut in our dataset, requiring about eight times less CPU time. For instances with a small number of nodes, arcs or commodities Gurobi returns better average gaps, at the expense, however, of larger CPU times. For larger instances, our algorithm outperforms branch-and-cut both in terms of CPU time and optimality gaps.

In a final experiment, we attempt to tackle instances in which branch-and-cut failed to return an optimality gap. There are 51 instances, all of which have 20 nodes and 100 commodities. For these instances, we tune our heuristics so that they spend more time searching for high quality feasible

solutions. Specifically, we (i) increase the time limit of the single-period PEP solved in line 4 of Algorithm 1 from 100 seconds to 500 seconds, (ii) increase the time limit of the fixing heuristic from 1,000 seconds to 2,000 seconds and (iii) we only employ the incremental heuristics and the fixing heuristic once at the end of Lagrange relaxation instead of applying the fixing heuristic twice and the local branching heuristic once. Columns (8) and (9) report the gap and CPU time of our algorithm for these instances. As evidenced, these 51 instances are particularly hard to tackle, with our algorithm obtaining an average gap of less than 10% and consuming more than one hour of CPU time. Although it is beyond the scope of our current research, an exact branch-and-bound algorithm based on Lagrange relaxation can be developed to tackle those instances and further reduce their optimality gap. The relatively short amount of CPU time required by Lagrange relaxation, and the fact that branching on a single arc is unlikely to change significantly the neighborhood of the optimal Lagrange multipliers, make such an approach a promising direction for future research.

C.2. Uncapacitated Instances.

We investigate the efficiency of our Benders decomposition implementations for uncapacitated network design instances by utilizing the same set of multi-period instances but without considering their capacity. Specifically, we benchmark (i) a basic implementation (B-REG); (ii) Adding PO cuts (B-MW1); (iii) Adding PO cuts solving only one subproblem, as in Magnanti et al. (1986) (B-MW2) and finally the formulation of Fischetti et al. (2010) (B-F). All implementations update the core point using Remark 3, and exploit modern callback technology to avoid solving the master problem multiple times (Bai and Rubin 2009) and make use of the cutting planes described in section 5.2.5. If a primal subproblem is infeasible, all methods except (B-F) generate a feasibility cut of the form (23) by using the extreme rays of the dual subproblem (19)-(21). Similar to the capacitated experiments, our basic benchmark is the best Gurobi (GRB) formulation, which uses the cut pool to handle the separation of the strong inequalities (18). Table 14 shows the average optimality gaps and CPU times obtained by each method, respectively.

A careful analysis of Table 14 suggests some important observations. First, Benders reformulations seem to be intrinsically more efficient compared to branch-and-cut (GRB). Specifically, our best implementation (B-F), attains an average gap which is more than one order of magnitude lower, and consumes about 60% of GRB's CPU time. This performance difference has been observed for other problems as well, such as facility location (Fischetti et al. 2016), suggesting that a modern implementation of Benders decomposition can be a superior alternative to an off-the-shelf, state-of-the-art solver. Second, with respect to the basic Benders implementation, we note that although it achieves

		Optimality Gap (%)					CPU Time (s)				
$\|\mathcal{N}\|$	#	GRB	B-REG	B-MW1	B-MW2	B-F	GRB	B-REG	B-MW1	B-MW2	B-F
10	216	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	76	72	275	23	**15**
20	216	30.24	12.93	4.66	3.04	**0.83**	3,729	4,310	5,468	3,144	**3,129**
$\|\mathcal{A}\|$											
35	72	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	34	47	78	17	**11**
60	72	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	75	61	395	20	**14**
83	72	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	119	108	353	31	**19**
120	72	12.84	1.72	0.47	0.16	**0.00**	1,984	4,107	4,342	**1,530**	1,535
220	72	32.27	16.89	2.65	1.35	**1.26**	4,439	**3,910**	5,989	3,940	4,002
318	72	45.61	20.18	10.87	7.61	**1.24**	4,763	4,913	6,083	3,963	**3,851**
$\|\mathcal{K}\|$											
50	216	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	76	72	275	23	**15**
100	216	30.24	12.93	4.66	3.04	**0.83**	3,729	4,310	5,468	3,144	**3,129**
T											
20	108	3.18	1.92	0.59	0.19	**0.00**	1,113	2,131	2,232	956	**941**
40	108	12.84	3.54	1.2	0.66	**0.29**	1,902	2,473	2,905	1,599	**1,592**
60	108	22.3	7.83	1.6	0.86	**0.48**	2,208	2,251	3,367	**1,971**	1,998
80	108	22.16	12.57	5.94	4.38	**0.90**	2,387	1,909	2,953	1,807	**1,756**
Total	432	15.12	6.47	2.33	1.52	**0.42**	1,902	2,191	2,866	1,583	**1,572**

Table 14 Optimality gaps and CPU times of Gurobi and Benders.

a far better gap than branch-and-cut, it falls behind the more sophisticated implementations by a large margin, implying that the impact of adding PO cuts significantly enhances performance. In addition to the poor CPU time performance of basic Benders, it is worth noting that it ran out of memory in 36 instances. These 36 instances consumed an average of 1,787 seconds of CPU time before running out of memory and resulted in an average gap of 55%, which is still better compared to Gurobi's average gap for these instances, which is 85%. Third, when assessing the impact of PO cuts, we observe a non-trivial gap improvement between the basic Benders (B-REG) and the basic PO Benders (B-MW1) implementations. Further, (B-MW1) ran out of memory in 12 instances instead of 36, but these improvements come at a high CPU time cost, because two LPs are needed in order to generate every cut, making (B-MW1) have the worst time performance across all meth-

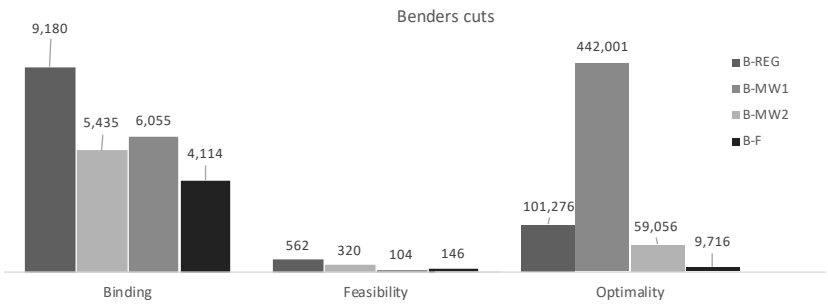


Figure 3 **Benders cuts generated by each implementation. Optimality and feasibility refer to the total number of each type generated during branch-and-cut. Binding are the cuts binding at an optimal solution.**

ods. Fourth, implementation (B-MW2), which generates PO cuts using a single subproblem per iteration dominates (B-MW1) in terms of both gap and CPU time performance, across every single averaged configuration, while it ran out of memory for six instances. This confirms the importance of an efficient generation of PO cuts. Fifth, (B-F) is the best-performing implementation, having both the lowest gap and CPU time, while no instances ran out of memory. In particular, it was able to close the gap for 54 instances that GRB failed to solve to optimality, and for 39 instances that no algorithm solved optimally, nine of which have approximately 2.5 million variables. The ability of this model to incorporate the current flow cost in the cut-generating subproblem and to generate cuts that unify feasibility and optimality leads to important improvements over the Magnanti-Wong implementations. Finally, the breakdown per instance attribute shows that problems with 10 nodes and 50 commodities can all be solved to optimality rather easily by all algorithms, while problems with 20 nodes and 100 commodities are far more challenging. Problems with more arcs or periods are also more challenging to solve, while the toughest instances tend to be the largest ones, 18 of which have 2.5 million variables.

On further analysis, it is interesting to investigate the strength of the cuts each Benders implementation requires. To this end, Figure 3 shows the average number of binding, optimality and feasibility cuts each algorithm generated in a subset of instances in which all algorithms could find an optimal solution.

Figure 3 suggests some important conclusions. First, (B-F) has the smallest number of cuts binding at an optimal solution, suggesting that it generates cuts of high quality. Second, only a limited number of feasibility cuts are added per instance. This is expected, since these instances are generally dense, and we already add origin-destination cuts, which are necessary for feasibility. However, this is in sharp contract with the Pazour instances, where the networks are much sparser,

and all methods generate an order of magnitude more feasibility cuts. When it comes to optimality cuts, perhaps what is the most surprising is that (B-MW1) generates a very large number of them. This pattern is also observed for the Pazour instances, suggesting that a textbook implementation of Magnanti-Wong cuts may result in generating a large number of them. Nevertheless, the number of cuts binding at an optimal solution is lower than that of (B-REG) and (B-MW2).