Incremental Network Design with Multi-commodity Flows

Ian Herszterg*and Martin Savelsbergh†

Georgia Institute of Technology

H. Milton Stewart School of Industrial and Systems Engineering
755 Ferst Drive, NW, Atlanta, GA 30332, U.S.

Abstract

We introduce a novel incremental network design problem motivated by the expansion of hub capacities in package express service networks: the *incremental network design problem with multi-commodity flows*. We are given an initial and a target service network design, defined by a set of nodes, arcs, and origin-destination demands (commodities), and we seek to find a transition from the initial service network to the target service network. In the target service network design, the capacity of a subset of arcs has been increased (hub capacities can be modeled as arcs in the network). In each period, the capacity of a single arc can be increased and the cost in a period is given by the solution to an unsplittable multi-commodity flow problem. Our objective is to find a sequence of arc capacity expansions such that the total cost during the transition is minimized. We model the problem as an integer program, propose and analyze greedy heuristics and develop an exact solution approach. We provide worst-case analyses for the greedy heuristics and compare the efficacy of the algorithms to solving the integer programming formulation of the problem using a commercial solver.

keywords: Incremental optimization; Capacity planning; Transportation.

1 Introduction

Less-than-truckload and package express transportation carriers regularly evaluate the design and operations of their service network. This can be prompted by a change in (forecast) demand or simply by a desire to identify cost savings (e.g., in the shuttle and line haul transportation costs or in the operational cost at hubs). However,

*Corresponding author (iherszterg@gatech.edu).
†martin.savelsbergh@isye.gatech.edu

changing the design and operations of an existing service network is not trivial and typically proceeds in two steps. First, planners decide where upgrades or expansions are introduced (e.g., where additional hubs will be located and where sorting capacity of existing hubs will be increased). Second, planners decide when to implement these upgrades and expansions. Given that a service network redesign may result in significant changes in the operations, the transition to a new target design typically occurs gradually, in a number of phases. Another reason for a gradual transition is that the change of the service network may require significant investments and budgets may force these investments to be made over a period of time. Given that there will be a gradual transition from an existing service network to a target service network, it becomes important to properly sequence the changes as this sequence directly affects the operational costs and the profit during the entire transition period. In incremental service network design problems, we assume that an initial service network design and a target service network design are provided, and we seek, in a fixed number of steps, to transition from the initial service network design to the target network design, where the goal is to maximize or minimize an objective function defined over the entire transition period (e.g., we want maximize the sum of the profits made in each of the transition periods).

In this paper, we introduce a novel incremental network design problem motivated by the expansion of hub capacities in package express service networks: the *incremental network design problem with multi-commodity flows*. We are given an initial and a target service network design, defined by a set of nodes, arcs, and origin-destination demands (commodities), and we seek to find a transition from the initial service network to the target service network. In the target service network design, the capacity of a subset of arcs has been increased (hub capacities can be modeled as arcs in the network). In each period, the capacity of a single arc can be increased and the cost in a period is given by the solution to an unsplittable multi-commodity flow problem modeling the transportation costs in the network. Our objective is to find a sequence of arc capacity expansions such that the total cost during the transition is minimized.

We model the problem as an integer program, propose and analyze greedy heuristics and develop an exact solution approach. In the greedy heuristics, we determine the sequence of expansions by selecting arcs based on: (1) the size of expansion (static), and (2) the largest immediate reduction in flow cost (dynamic). For finding optimal solutions, we propose a depth-first search partial enumeration algorithm that explores partial sequences of expansions in a depth-first search manner, where, for each partial sequence, we compute: (1) the cost associated with the partial sequence, and (2) upper and lower bounds on the costs of the remaining transition periods, which are used to curtail the enumeration of all partial sequences. We provide worst-case analyses for the greedy heuristics and compare the efficacy of the algorithms to solving the integer programming formulation of the problem using a commercial solver. The contributions of this research are summarized as follows:

- A new incremental network design problem is introduced, motivated by multi-phase hub capacity expansion in package express service networks;
- A number of greedy heuristics and an exact algorithm are designed, imple-

mented, and analyzed, and;
- A computational study using instances derived from real-world data of a large package express carrier is conducted;

The remainder of this paper is organized as follows. In Section 2, we review relevant prior research on incremental network design and capacity expansion problems. In Section 3, we provide a formal description of (general) incremental network design problems and of the incremental network design problem with unsplittable multi-commodity flows. In Sections 4 and 5, we propose greedy heuristics to quickly obtain high quality solutions and an exact approach to obtain optimal solutions, respectively. In Section 6, we present and interpret the results of an extensive computational study. Finally, in Section 7, we finish with final remarks.

2 Literature Review

Since we are not aware of any literature addressing the incremental network design problem with multi-commodity flows, we briefly review literature on capacity expansion problems and on incremental network design problems.

There is a wide variety of capacity expansion problems covered in the literature for different types of service networks. For example, in the context of telecommunication networks, Gendreau et al. (2006) proposes a heuristic approach for finding the least cost alternative of installing additional concentrators at the nodes and cables on the links of the network in order to meet an increasing demand. In the context of water distribution systems, Hsu et al. (2008) proposes capacity expansion alternatives to an existing water distribution system in order to improve the efficiency of water supply. For large urban transportation networks, Mathew and Sharma (2009) and Marin and Jaramillo (2008) propose different strategies for opening new links and adding capacity to existing links in the network in order to minimize congestion. In gas transportation networks, Andre et al. (2009) presents techniques for finding the optimal location of pipeline segments to be expanded and the optimal size of these expansions such that investment costs on an existing gas transportation network are minimized. There is also a fair amount of literature purely focused on facility expansion problems concerned with the timing of facility expansions to meet increasing demand (see, e.g. Jablonowski et al. (2011) and Wang and Lin (2002)). These problems, however, are focused almost exclusively on finding the set of necessary expansions/changes in the service network design in order to meet the desired goals, and do not investigate how to gradually transition from the current state of the network to a provided target network design. For a more broad survey on capacity expansion problems covered in the of field Operations Research, see Luss (1982).

Incremental network design problems, which integrate both network design and scheduling problems, were originally introduced in Nurre et al. (2012) and Jeong and Culler (2004). In the context of electrical power grids, there is a fair amount of literature on incremental network design problems where the objective is to convert the current design of an electrical power grid into a smart grid, where resource and budget constraints allow only a limited number of upgrades per time period (see, e.g.

DeBlasio and Tom (2008) and Farhangi (2009)). Another example of incremental network design problems often studied in the literature is the one faced by managers of critical civil interdependent infrastructure systems of restoring essential public services after a non-routine event causes disruptions in the system. In this context, the objective is to determine a set of tasks to be completed, assign these tasks to work groups, and then determine the schedule of each work group to complete the tasks assigned to it such that a quality measure over the entire horizon of the restoration plan is maximized (see, e.g. Cavdaroglu et al. (2013) and Çelik (2016)).

In a setting more similar to the one we consider, Kalinowski et al. (2015) studies an incremental network design with maximum flows problem where, given a current network design defined over a set of nodes and arcs with fixed integer capacity, the goal is to build new arcs in the network from a set of fixed potential arcs, one at a time, such that the maximum flow in the network increases and the cumulative performance, i.e., the sum of the performance measures of the networks in all time periods, is maximized. The authors propose different integer programming formulations for its solution and a set of greedy heuristics to find a good sequence of arcs to build in the network and meet the desired goal. The worst-case analyses for the greedy heuristics are also presented. Other incremental network design problems where, in each period, one or more arcs from a set of potential arcs are built in the network and classical network problems have to be solved with the overall goal of minimizing or maximizing performance metrics over the entire transition period include: incremental network design with shortest paths Baxter et al. (2014) and incremental network design with minimum spanning trees Engel et al. (2017). In these settings, the network optimization problems solved in each period are all polynomially solvable. However, the incremental network design version may be NP-complete (e.g., Baxter et al. (2014) and Nurre and Sharkey (2014)). When a single arc is built in each period, the authors in Engel et al. (2017) show that the incremental network design problem with minimum spanning trees can be solved efficiently using a greedy heuristic.

3 Problem Statement

We start by illustrating the concepts of incremental network design problems mathematically. Let x be a decision vector representing a service network design (e.g., the location and type of facilities, the shipment flow paths, etc.). Let $p \cdot x$ represent the profit of a design x and let $Ax \leq b$ represent feasibility constraints (i.e., constraints on the service network design). Finally, let x^0 and x^T represent an initial and a target network design, respectively, and let T be the number of transition periods (e.g., we want transition from an initial design to a target design in T quarters, or if a target network design has T additional hubs, we want to add the new hubs one by one). Then, we are seeking intermediate network designs x^i such that

$$x^0 \to x^1 \to x^2 \to \cdots \to x^{T-1} \to x^T.$$

Each step $x^{j-1} \to x^j$ of the transformation process corresponds to a separate design problem with constraints

$$Ax^j \leq b$$
$$|x^j - x^{j-1}| \leq \Delta,$$

where the constraint $|x^j - x^{j-1}| \leq \Delta$ reflects that only limited adjustments to design x^{j-1} can be made. Importantly, the objective covers the entire transition period, i.e.,

$$\max \sum_{i=1}^{T-1} p \cdot x^i$$

This captures the fact that we want to maximize the profit over the entire transition period, which is not necessarily the same as maximizing the profit in each individual transition period. By maximizing the increase in profit in a single period, we may put ourselves in a position that makes it difficult to achieve additional profit in the next period (because we are restricted in the changes we can make to the design in each period). Additional constraints can be added to the model representing company goals/policies, e.g., $p \cdot x^j \geq p \cdot x^{j-1}$ (profits are not allowed to decrease in any step).

Next, we provide a formal description of the incremental network design problem with multi-commodity flows. Let x^0 be the initial design representing a multi-commodity flow solution for a commodity set K to be served on a directed network $G^0 = (N, A)$ with node set N and arc set A. For each arc $a \in A$, $c_a \in \mathbb{R}_+$ represents the cost of sending one unit of flow through the arc and $u_a \in \mathbb{Z}_+$ is the (integer) capacity of the arc. Furthermore, let x^T be the target design, representing a multi-commodity flow solution for the same commodity set K on the directed network $G^T = (N, A)$ with the same set of nodes and arcs as G^0, but with a subset of arcs $\hat{A} \subseteq A$ with expanded capacity $u_a + w_a$, where $w_a \in \mathbb{Z}_+$ represents the additional capacity of arc $a \in \hat{A}$. For each $k \in K$, let $o_k \in N$ denote the commodity's origin, let $d_k \in N$ denote the commodity's destination, and let $q_k \in \mathbb{R}_+$ denote that quantity that needs to be routed from the origin to the destination of the commodity k along a single path (i.e., we consider the unsplittable variant). We assume that $q_k \leq u_a$ for all $k \in K$. Finally, let $T = |\hat{A}|$ be the length of the transition period.

We will seek for the optimal transition of the initial network design to the target network design in T steps where, in each step, we expand the capacity of exactly one arc in $\hat{A}$ and record the multi-commodity flow costs in the modified network. The objective is to minimize the total multi-commodity flow costs over the transition period:

$$z^* = \min \sum_{t=1}^{T-1} z(x^t) \tag{1}$$

where $z(x^t)$ denotes the multi-commodity flow cost in the intermediate design at period t. Note that we do not consider the multi-commodity flow costs in the initial and target network designs since these are sunk costs.

The problem can be formulated as an integer program. Let $x^t_{a,k}$ be a binary decision variable representing sending all units from a commodity $k \in K$ through the arc $a \in A$ at a time period t ($x^t_{a,k} = 1$) or not ($x^t_{a,k} = 0$). Let y^t_a be another binary decision variable indicating that the arc $a \in \hat{A}$ has its capacity increased by w_a units at time t ($y^t_a = 1$) or not ($y^t_a = 0$). We formulate the incremental network design problem with multi-commodity flows as the following integer program:

$$\text{(INCMCF)} \quad \min \sum_{t=1}^{T-1} \sum_{a \in A} \sum_{k \in K} (q_k \cdot c_a) \cdot x^t_{a,k}$$

$$\sum_{a \in \delta^+(i)} x^t_{a,k} - \sum_{a \in \delta^-(i)} x^t_{a,k} = \begin{cases} 1, & i = o_k \\ -1, & i = d_k \\ 0, & \text{o.w.} \end{cases} \quad \forall i \in N, k \in K, t = 1, \ldots, T-1 \tag{2}$$

$$\sum_{k \in \mathcal{K}} q_k \cdot x^t_{a,k} \leq u_a, \quad \forall t = 1, \ldots, T-1, \ a \in A \setminus \hat{A} \tag{3}$$

$$\sum_{k \in \mathcal{K}} q_k \cdot x^t_{a,k} \leq u_a + y^t_a \cdot w_a, \quad \forall t = 1, \ldots, T-1, \ a \in \hat{A} \tag{4}$$

$$y^t_a \leq y^{t+1}_a, \quad \forall t = 1, \ldots, T-2, \ a \in \hat{A} \tag{5}$$

$$\sum_{a \in \hat{A}} y^t_a = t, \quad \forall t = 1, \ldots, T-1 \tag{6}$$

$$x^t_{a,k} \in \{0, 1\}, \quad \forall t = 1, \ldots, T-1, \ a \in A, \ k \in K \tag{7}$$

$$y^t_a \in \{0, 1\}, \quad \forall t = 1, \ldots, T-1, \ a \in \hat{A} \tag{8}$$

Constraints (2) and (3) are the typical integer multi-commodity flow constraints, where the flow assignment decision variables are also indexed by time period. For every arc $a \in \hat{A}$ and time period t, Constraints (4) ensure that the capacity of the arc is increased by w_a units if the arc is selected for expansion at or before time period t. Constraints (5) ensure that if the capacity of the arc $a \in \hat{A}$ is selected for expansion by time t ($y^t_a = 1$), then all decision variables $y^{\bar{t}}_a$ for $t < \bar{t} \leq T-1$ will also be set to one (i.e. $y^{t+1}_a = 1, y^{t+2}_a = 1, \ldots, y^{T-1}_a = 1$). Constraints (6) ensure that exactly one arc $a \in \hat{A}$ is selected for expansion at each time period. The objective function aims to minimize the total transition costs from the initial network design to the target network design, where the associated cost for each time period t is given by the solution of a integer multi-commodity flow problem. The integrality constraints of the $x^t_{a,t}$ decision variables (constraint 7) ensure unsplittable flows for commodities. We denote this formulation by INCMCF.

Proposition 1. *Let $z(x^0)$ and $z(x^T)$ be the multi-commodity flow costs of the initial and target network designs, respectively, and let z^* be the optimal transition costs from the initial to the target network design (representing the $T-1$ intermediate transition periods). Then, $(T-1)z(x^T) \leq z^* \leq (T-1)z(x^0)$.*

Proof. Given that $z(x^t)$ is a monotonic non-increasing function with respect to the multi-commodity flow costs in the intermediate designs along consecutive transition periods (which follows from $w_a \in \mathbb{Z}_+$), then the lowest and highest possible values for $z(x)$ are achieved in x^T and x^0, respectively. Thus, both sides of the inequality represent trivial lower and upper bounds for z^*. □

4 Greedy Heuristics

4.1 By size of expansion

Expanding the capacity of arcs by size of expansion, from highest to lowest, is a natural greedy strategy (GreedySizeExp-HL). A large increment in capacity suggests that more commodity paths can benefit from using the expanded capacity, which can reduce the current flow costs, but also future flow costs (i.e., in subsequent transition periods). We start by sorting the arcs in $\hat{A}$ by size of expansion, from highest to lowest. For each transition period t, we expand the capacity of a single arc in the network following the order in the sorted list and re-compute the multi-commodity flow costs in the modified network. That is, we solve $T-1$ multi-commodity flow problems, one for every transition period. The greedy heuristic is described in Algorithm 1. GreedySizeExp-HL break ties by selecting the arc with largest number of commodity paths using the arc and, if that does not resolve a tie, by selecting an arc at random (with equal probability).

Algorithm 1 GreedySizeExp-HL

input:$\hat{A}$ - set of candidate arcs for expansion
output: $\bar{c}$ - upper bound on the total transition costs

$\bar{c} \leftarrow 0$
Sort $\hat{A}$ by size of expansion, from highest to lowest
for each transition period $t = 1, \ldots, T-1$ **do**
  $\hat{a} \leftarrow \{\hat{A}_t\}$
  $u_{\hat{a}} \leftarrow u_{\hat{a}} + w_{\hat{a}}$
  $\bar{x} \leftarrow$ SolveMCF$(\cdot)$
  $\bar{c} \leftarrow \bar{c} + z(\bar{x})$
end for
return $\bar{c}$

Figure 1a shows an example of an instance representing the worst case scenario for GreedySizeExp-HL. In the figure, the cost of each arc is shown in black, the original capacity of each arc is shown in blue, and the size of the expansion for the three arcs to be expanded is shown in red (i.e., for arcs AB, BC and CD). There are three commodities with unit demand and origin-destination pairs: AD, BD, and CD. The initial and target network designs are depicted in Figures 1b and 1c, with flow costs $3L + 7\epsilon$ and $L + 3\epsilon$, respectively (the arcs with flow are shown in green). Increasing the capacity of arcs by size of expansion, from highest to lowest (CD, then

BC, then AB), yields a total transition cost of $(3L + 5\epsilon) + (3L + 3\epsilon) = 6L + 8\epsilon$. The cheapest transition costs, however, is to expand the capacity of arcs from lowest to highest size of expansion (AB, then BC, then CD), resulting in a total cumulative flow cost of $(L + 9\epsilon) + (L + 6\epsilon) = 2L + 15\epsilon$ (we assume $L \gg 3\epsilon$). This example can be generalized to a family of worst case instances for larger values of T (and larger sets K) by adding new small rectangles of side ϵ to the base of the outer rectangle of size L as shown in in Figure 2. Let z_{GHL} be the total cumulative flow costs by expanding arcs by size of expansion, from highest to lowest. We can express $z(x^0)$, $z(x^T)$ and z_{GHL} in terms of T, L and ϵ, as follows:

$$z(x^0) = 3L + \sum_{k=1}^{T-1}(k+2)\epsilon \tag{9}$$

$$z(x^T) = L + \sum_{k=1}^{T-1} k\epsilon \tag{10}$$

$$z_{GHL} = (T-1)\cdot(3L + \sum_{k=1}^{T-1} k\epsilon) + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \tag{11}$$

Proposition 2. *In the worst case,* GreedySizeExp-HL *gets arbitrarily close to* $(T-1)z(x^0)$.

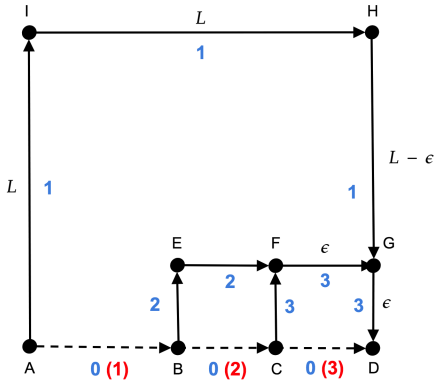
Proof. Consider the family of instances shown in Figure 2. By Proposition 1, we have that $z^* \leq z_{GHL} \leq (T-1)z(x^0)$. From (9) and (11) we obtain:

$$\begin{aligned}
z_{GHL} &= (T-1)\cdot(3L + \sum_{k=1}^{T-1} k\epsilon) + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \\
&= (T-1)\cdot(3L + \sum_{k=1}^{T-1}(k+2)\epsilon - \sum_{k=1}^{T-1} 2\epsilon) + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \\
&= (T-1)\cdot(3L + \sum_{k=1}^{T-1}(k+2)\epsilon) - (T-1)\sum_{k=1}^{T-1} 2\epsilon + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon \\
&= (T-1)z(x^0) - \underbrace{(T-1)\sum_{k=1}^{T-1} 2\epsilon + \sum_{k=1}^{T-1}\sum_{i=1}^{k-1} 2\epsilon}_{=\gamma\epsilon} \\
&= (T-1)z(x^0) - \gamma\epsilon \\
&\leq (T-1)z(x^0)
\end{aligned}$$

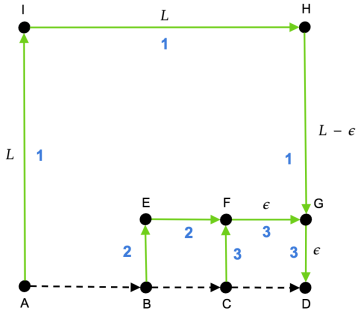
Finally, when ϵ goes to zero:

$$\begin{aligned}
\lim_{\epsilon\to 0} z_{GHL} &= \lim_{\epsilon\to 0} (T-1)z(x^0) - \gamma\epsilon \\
&= (T-1)z(x^0)
\end{aligned}$$

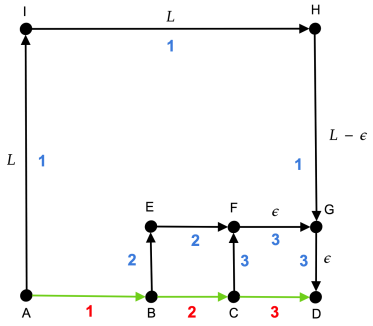
□



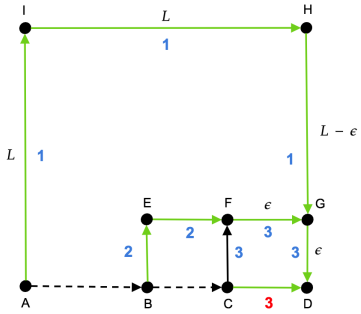
(a) Worst case example for GreedySizeExp-HL



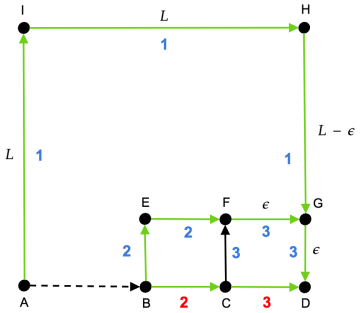
(b) Initial network design



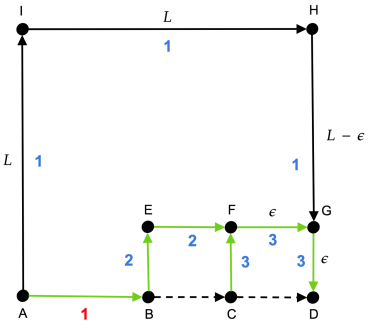
(c) Target network design



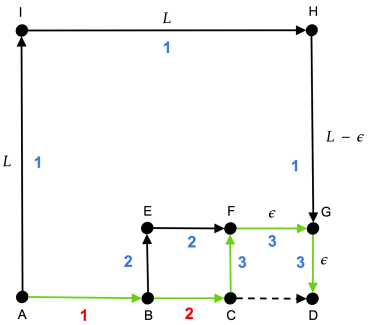
(d) First intermediate design using GreedySizeExp-HL



(e) Second intermediate design using GreedySizeExp-HL



(f) First intermediate design in the optimal solution



(g) Second intermediate design in the optimal solution

Figure 1: Worst case example for GreedySizeExp-HL with three origin-destination pairs with unit demand each (AD, BD and CD), showing the initial and target network designs and the intermediate designs obtained using the greedy heuristic and in the optimal solution.

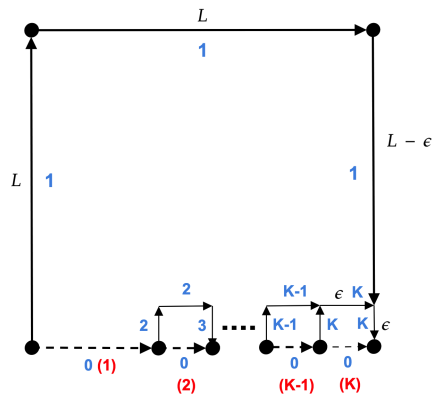


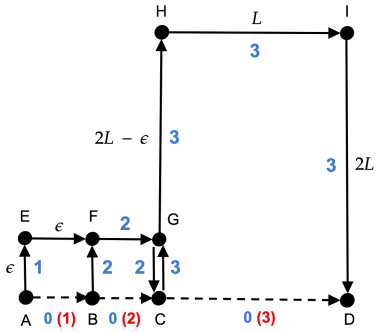
Figure 2: Family of worst case instances for GreedySizeExp-HL

Figure 3 shows an example of the worst case scenario for choosing arcs for expansion in the opposite sorting direction, from lowest to highest size of expansion, which we do in GreedySizeExp-LH. We consider again three different origin-destination pairs, with unit demand each (AD, BD and CD). Using a proof similar to the one for Proposition 2, we can show GreedySizeExp-LH also achieves a tight upper bound on the transition costs z_{GLH} of $z_{GLH} \leq (T-1)z(x^0)$ in the worst case when ϵ goes to zero.

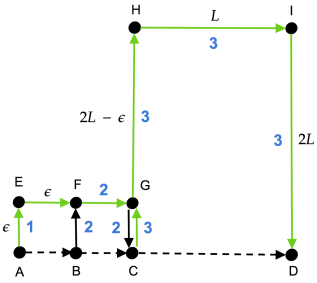
4.2 By largest reduction in flow costs

As we have seen, a weakness of GreedySizeExp is that there can be periods in which there is little or no reduction in the period flow costs. In order to overcome this weakness, we next explore a second greedy heuristic that focuses on the immediate reduction in flow costs rather than the size of the capacity expansion (GreedyCostRed). The main idea is to identify, for each arc for which we can expand the capacity, a set of commodities that are likely to benefit from being able to use the added capacity and then choose the arc that yields the (potential) largest anticipated benefit.

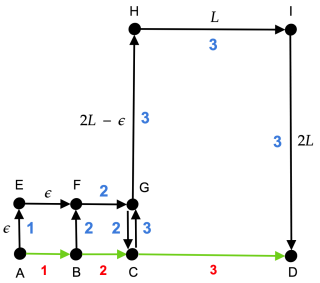
More specifically, for each transition period t, let $\hat{A}^t$ be the subset of arcs that is still available for expansion at period t, $p_{k,t-1}$ be the path assigned to a commodity $k \in K$ at period $t-1$ and $f_{a,t-1}$ be the flow going through a at period $t-1$. Furthermore, let $p^*_{k,a}$ be the shortest path connecting the origin to the destination of commodity k that uses arc a. For each candidate arc $a \in \hat{A}^t$ and commodity $k \in K$, we start by computing the difference between the cost of $p_{k,t-1}$ and the cost of $p^*_{k,a}$, i.e., $r_{k,a,t} = [c(p_{k,t-1}) - c(p^*_{k,a})]_+$. Once the values $r_{k,a,t}$ are computed for all commodities, we compute for each arc a the sum $s_{a,t} = \sum r_{k,a,t}$, starting from the highest to lowest values of $r_{k,a,t}$ until the extra capacity of w_a units has been consumed (see Algorithm 2). We then choose for expansion the arc $\hat{a} = \arg\max_{a \in \hat{A}^t} s_{a,t}$, which indicates that augmenting the capacity of $\hat{a}$ potentially yields the largest immediate reduction in flow costs. Finally, before moving on to the next transition period, we re-compute the multi-commodity flow costs in the updated network. GreedyCostRed break ties by selecting arcs with the highest size of expansion first, then by largest number



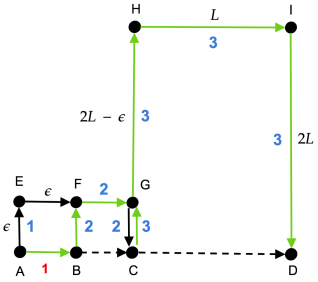
(a) Worst case example for GreedySizeExp-LH



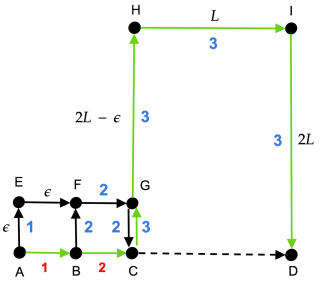
(b) Initial network design



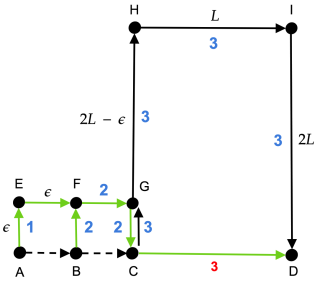
(c) Target network design



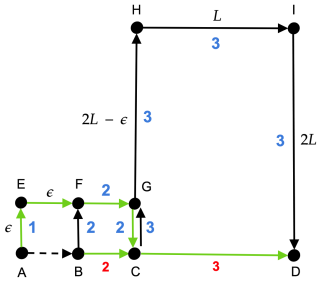
(d) First intermediate design using GreedySizeExp-LH



(e) Second intermediate design using the GreedySizeExp-LH



(f) First intermediate design in the optimal solution



(g) Second intermediate design in the optimal solution

Figure 3: Worst case example for GreedySizeExp-LH with three origin-destination pairs with unit demand each (AD, BD and CD), showing the initial and target network designs and the intermediate designs obtained using the greedy heuristic and in the optimal solution. Arcs used by commodity paths are highlighted in green.

of commodity paths using the arc, and finally by randomly selecting an arc. Shortest paths are pre-computed using Floyd-Warshall's algorithm in $\mathcal{O}(|N|^3)$ at the beginning of the algorithm and stored in a table for quick access.

Figures 4 and 5 illustrate the first iteration of GreedyCostRed on the worst case instances for GreedySizeExp, showing that the algorithm makes the optimal selection for the first arc to expand in both cases. Figure 6, however, shows an example where GreedyCostRed fails to find the optimal transition sequence. We consider again three different origin-destination pairs, with unit demand each (K=3, AH, BF and LM) and three candidate arcs for expansion (T=3, AB, BG and LM). The initial and target network designs are depicted in Figures 6b and 6c, with flow costs $3L + 10\epsilon$ and $L + 4\epsilon$, respectively. Increasing the capacity of arcs based on the largest immediate reduction in flow costs (LM then BG) yields a total cumulative flow cost of $6L + 10\epsilon$. The cheapest transition sequence, however, for $L > 4\epsilon$, is to expand the capacity of arcs BG then AB, resulting in a total cumulative flow cost of $4L + 16\epsilon$.

Proposition 3. *In the worst case,* GreedyCostRed *gets arbitrarily close to* $(T-1)z(x^0)$.

Proof. If we generalize the instance presented in Figure 6 for an arbitrary number of transition periods such that we only see a large reduction in the flow costs within a multiple of L at the very last intermediate transition period, then, in each of the intermediate periods, the flow cost will decrease in multiples of ϵ. As ϵ goes to zero, the transition costs will heavily depend on L and we will see a marginal difference between the flow costs in the intermediate designs and in the initial network design, i.e. $z(x^t) \approx z(x^0)$ for all $t = 1, \ldots, T-1$. Therefore, in the worst case, GreedyCostRed will get arbitrarily close to $(T-1)z(x^0)$. □

5 Exact method

Next, we propose a depth-first search algorithm with partial enumeration (DFSPE) for finding the optimal transition sequence of arc capacity expansions.

The algorithm enumerates transition sequences, exploring partial sequences of expansions in a depth-first search manner with a pre-specified order, where we compute, for each partial sequence: (1) the cost associated with the partial sequence, and (2) upper and lower bounds on the remaining cost of the transition sequence, used to curtail the enumeration of all possible transition sequences. The main difference between DFSPE and a standard IP-based branch and bound algorithm for solving INCMCF is that, in each iteration, DFSPE solves the unsplittable multi-commodity flow problem for a single transition period and decides which arc to expand next following a pre-defined order, rather than branching on a single integer variable and re-solving the model considering all transition periods at once. Furthermore, given that DFSPE only considers a single intermediate design at a time, it consumes far less memory than branch and bound algorithms used in commercial solvers. This is especially true

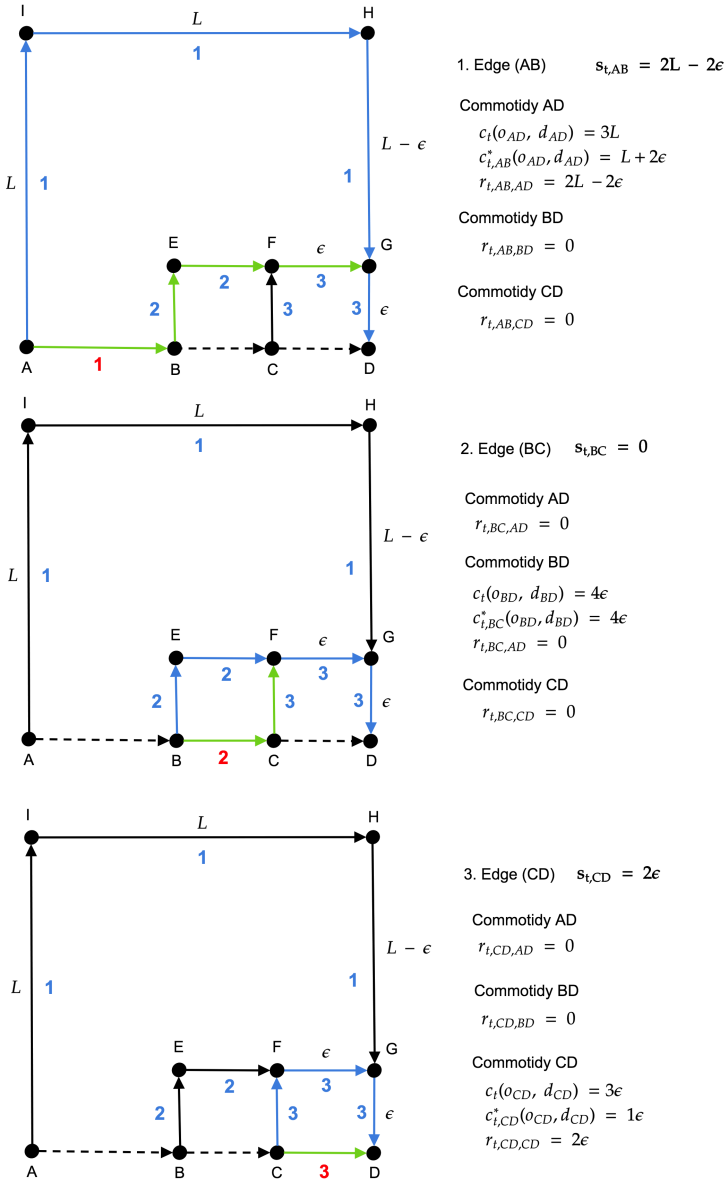


Figure 4: First iteration of GreedyCostRed on the worst case example for GreedySizeExp-HL, showing that the algorithm chooses arc AB to expand first. Commodities that benefit from the arcs in the sequence of expansions have their original paths highlighted in blue and the shortest paths using the expanded arcs in green. arcs belonging to both paths are kept in blue.

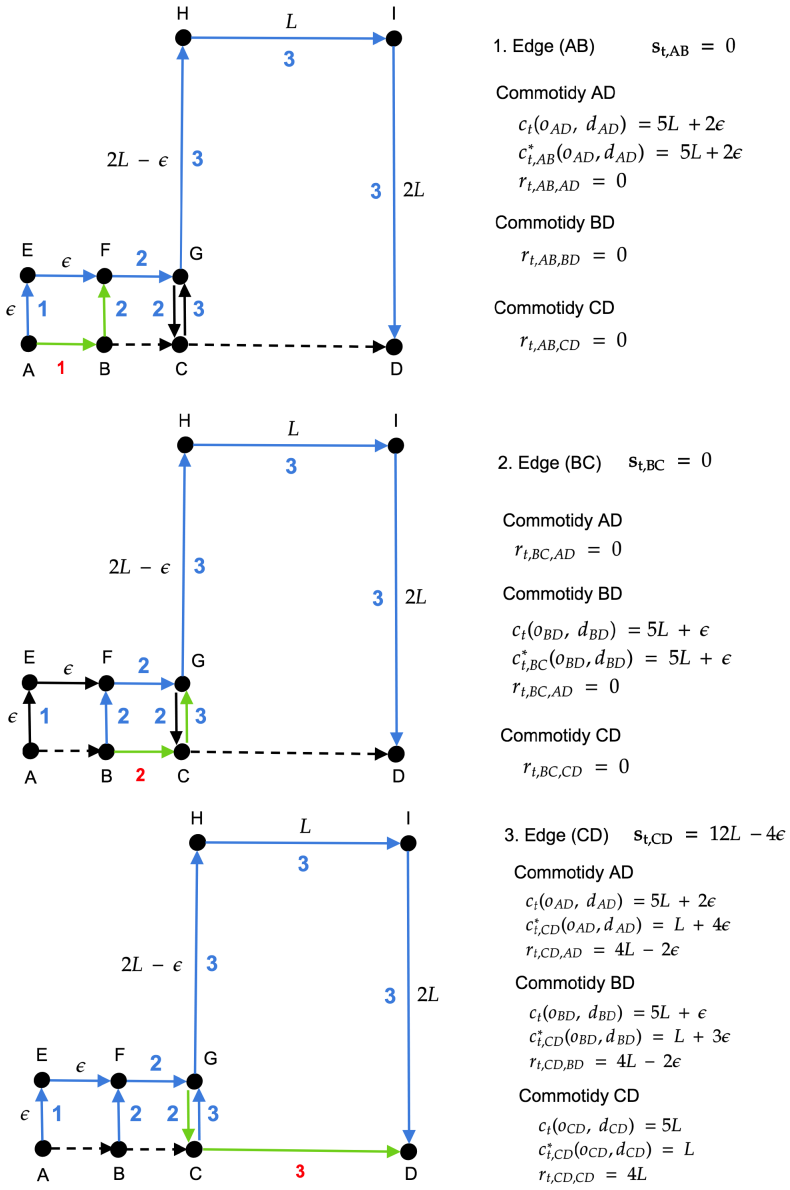
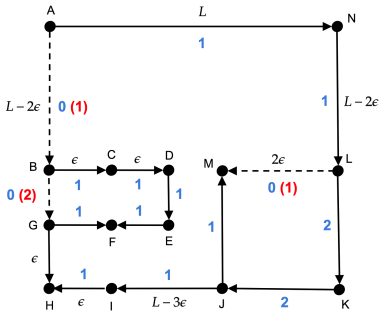
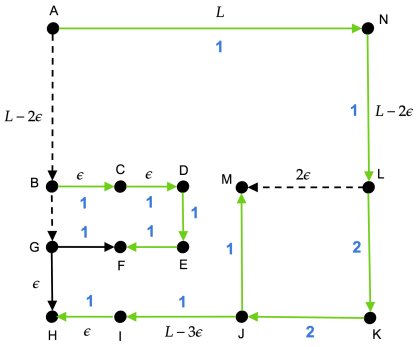


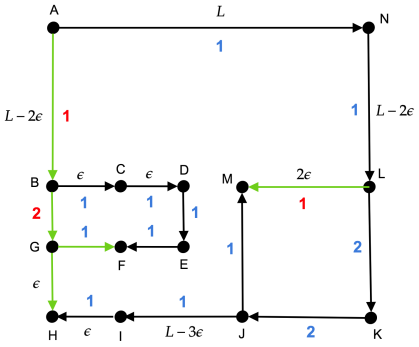
Figure 5: First iteration of GreedyCostRed on the worst case example for GreedySizeExp-LH, showing that the algorithm chooses arc CD to expand first. Commodities that benefit from the arcs in the sequence of expansions have their original paths highlighted in blue and the shortest paths using the expanded arcs in green. arcs belonging to both paths are kept in blue.



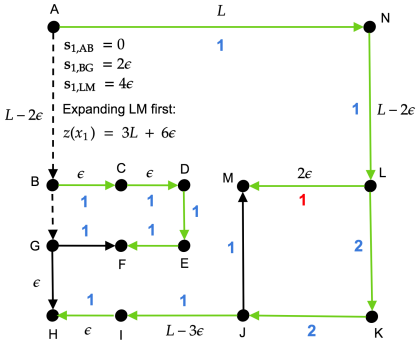
(a) Bad example for GreedyCostRed



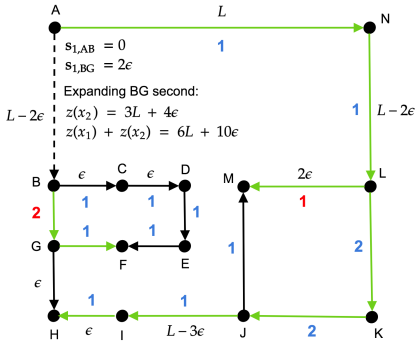
(b) Initial network design



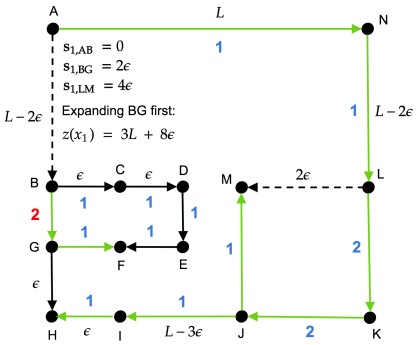
(c) Target network design



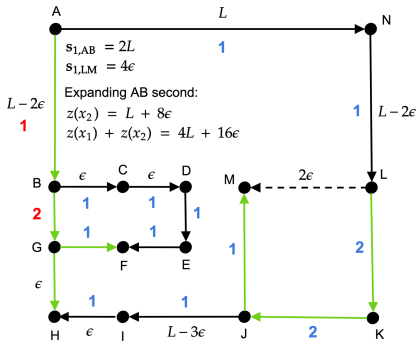
(d) First intermediate design using GreedyCostRed



(e) Second intermediate design using the GreedyCostRed



(f) First intermediate design in the optimal solution



(g) Second intermediate design in the optimal solution

Figure 6: Bad example for GreedySizeCostRed with three origin-destination pairs with unit demand each (AH, BF and LM) showing that the algorithm fails to find the optimal transition sequence. Arcs used by commodity paths are highlighted in green.

Algorithm 2 GreedyCostRed

input: $\hat{A}$ - set of candidate arcs for expansion
output: $\bar{c}$ - upper bound on the total transition costs

$\bar{c} \leftarrow 0$
$\hat{A}^1 \leftarrow \hat{A}$
for each transition period $t = 1, \ldots, T-1$ **do**
  for each arc $a \in \hat{A}^t$ **do**
    for each commodity $k \in K$ **do**
      $r_{k,a,t} \leftarrow [c(p_{k,t-1}) - c(p^*_{k,a})]_+$
    end for
    $s_{a,t} \leftarrow 0$
    $\bar{u}_a \leftarrow u_a + w_a - f_{a,t-1}$
    for each commodity $k \in K$, sorted by $r_{k,a,t}$, from highest to lowest, **do**
      if $\bar{u}_a - q_k \geq 0$ **then**
        $s_{a,t} \leftarrow s_{a,t} + r_{k,a,t}$
        $\bar{u}_a \leftarrow \bar{u}_a - q_k$
      end if
    end for
  end for
  $\hat{a} \leftarrow \arg\max_{a \in \hat{A}^t} s_{a,t}$
  $u_{\hat{a}} \leftarrow u_{\hat{a}} + w_{\hat{a}}$
  $\bar{x} \leftarrow \text{SolveMCF}(\cdot)$
  $\bar{c} \leftarrow \bar{c} + z(\bar{x})$
  $\hat{A}^{t+1} \leftarrow \hat{A}^t \setminus \{\hat{a}\}$
end for
return $\bar{c}$

when commercial solvers keep copies of the model and of the fractional solutions at every node in the branch and bound tree. In DFSPE, the multi-commodity flow solution on an intermediate design resulting from specific subsets of $\hat{A}$ are computed on-demand and stored in tables for quick look-up. We do so in order to avoid re-computing the flow costs for intermediate designs resulting from the same subset of expanded arcs (i.e. although the transition costs may differ, the multi-commodity flow solution for an intermediate design resulting, for example, from the sequence of expansions a_1, a_2 and a_3 is the same as the one resulting from the sequence a_3, a_1, a_2. That is, the multi-commodity flow solution is the same regardless of the order in which the arcs were chosen for expansion). Hence, during the execution of the algorithm, we only keep track of the flow costs on each intermediate design and use compact and sparse data structures to store the flow information on each arc.

As in branch and bound methods, DFSPE consists of a systematic enumeration of the candidate solutions for INCMCF. The set of complete sequences of expansions are encoded in the leaf nodes of a solution tree, where the cost associated to each node at a depth $D(i)$ from the root node in the tree represents the transition costs from periods $t = 1, \ldots, D(i)$ considering the partial sequence of arcs selected for

expansion in that branch. Nodes at a depth d in the tree serve as root for up to $T - d$ new branches, where each branch represents a different selection for the next arc to expand from the subset of $T - d$ remaining candidate arcs. When processing a node i at depth $D(i)$, we start, if necessary, i.e., when there is no entry in the table associated to that specific intermediate design, by computing the unsplittable multi-commodity flow costs on the intermediate design.

Before creating new branches rooted in node i, we compute lower and upper bounds on the transition costs for the remaining subsequent periods to decide if we should continue the search down that branch or not. Let $ub(i)$ and $lb(i)$ be lower and upper bounds on the costs for the $t = D(i) + 1, \ldots, T - 1$ remaining transition periods, respectively. Also, let $c(i)$ be the transition costs of the solution associated to the partial sequence encoded in node i and ub^* be the best upper bound on the optimal solution found so far. If $c(i) + lb(i) \geq ub^*$, then we stop the search in the current branch. If, on the other hand, $c(i) + lb(i) < ub^*$, then the next step is to return to P_i, the parent node of node i, select one of the remaining arcs to expand following the pre-defined order and continue the search in a depth-first manner. In order to do so, we choose the arc that potentially results in the largest immediate reduction in the multi-commodity flow costs, using the same procedure described in Section 4.2. The algorithm break ties by choosing the arc that yields the lowest upper bound on the remaining transition costs.

Whenever there are no more branches to explore in a given node, the algorithm then resumes the search from the parent node following the pre-specified order of arcs considered for expansion. The pseudo-code for the recursive implementation of DFSPE is shown in Algorithm 3. The heuristic procedures for obtaining upper and lower bounds are described next.

5.1 Computing upper and lower bounds

The effectiveness of DFSPE depends on the efficient determination of lower and upper bounds, both in terms of the quality and speed, which can lead to an exhaustive search in the solution tree when no branches of the tree are pruned. The challenge of obtaining fast high quality bounds is twofold: (1) choosing a good sequence of the remaining candidate arcs to expand, and (2) obtaining a fast and good approximation on the optimal solution for the unsplittable multi-commodity flow problem in the intermediate designs along the remaining transition periods. Next, we present fast heuristic approaches for obtaining such bounds.

5.1.1 Upper bounds

By assuming a specific remaining transition sequence of arcs selected for expansion, and by heuristically solving the multi-commodity flow problem in each of the intermediate designs, we obtain an upper bound on the cost of the remaining transition periods.

For each of the remaining transition periods $t = D(i)+1, \ldots, T-1$ in a node i, we will use the same policy for deciding the next arc to expand next, i.e. by the potential

Algorithm 3 Depth-first search algorithm with partial enumeration (DFSPE)

input: i - node index
$\hat{A}^i$ - set of remaining candidate arcs for expansion
$c(P_i)$ - solution cost at the parent node

if $|\hat{A}^i| = 1$ **then**
  if $c(P_i) < ub^*$ **then**
    $ub^* \leftarrow c(P_i)$
  end if
  return
else
  for each arc $a \in \hat{A}^i$ sorted by potential immediate reduction in flow costs, from highest to lowest **do**
    $u_a \leftarrow u_a + w_a$
    $\bar{x} \leftarrow$ SolveMCF$(\cdot)$
    $c(i) \leftarrow c(P_i) + z(\bar{x})$
    $ub(i) \leftarrow$ UB-H$(z(\bar{x}), \hat{A}^i \setminus \{a\})$
    $lb(i) \leftarrow$ LB-H$(\hat{A}^i \setminus \{a\})$
    if $c(i) + ub(i) < ub^*$ **then**
      $ub^* \leftarrow c(i) + ub(i)$
    end if
    if $c(i) + lb(i) < ub^*$ **then**
      $j \leftarrow$ create child node
      $DFSPE(j, \hat{A}^i \setminus \{a\}, c(i))$
    end if
    $u_a \leftarrow u_a - w_a$
  end for
  return
end if

largest immediate reduction in flow costs. Once an arc is selected for expansion, the next step is to obtain an upper bound on the multi-commodity flow costs in the resulting intermediate design.

For each commodity $k \in K$ and period t, let us consider again $p_{k,t-1}$ as the path assigned to k at period $t-1$, $p^*_{k,a}$ as the shortest path connecting the origin to the destination of k going through the arc a, and $f_{a,t-1}$ as the flow going through a at period $t-1$, where the arc a represents the arc selected for expansion at period t. Let us also consider p^*_k as the shortest path connecting the origin to the destination of k. The heuristic is divided in two parts: Phase 1 and Phase 2. In Phase 1, we first compute $r_{k,a,t} = [c(p_{k,t-1}) - c(p^*_{k,a})]_+$ for all commodities $k \in K$. We then start pushing commodities through $p^*_{k,a}$, starting from the highest to lowest values of $r_{k,a,t}$, whenever possible, until the expanded arc becomes saturated. When doing so, we always update the residual capacities of the arcs throughout the network. That is, a flow of size q_k is removed from all arcs in $p_{k,t-1}$ and added to all arcs in $p^*_{k,a}$.

Let $\bar{K}$ be the subset of K that contains the commodities that, either: (1) could

not fit in one or more arcs in $p^*_{k,a}$, or (2) satisfy $c(p_{k,t-1}) < c(p^*_{k,a})$. Note that when we successfully push commodities in cheaper paths using arc a in Phase 1, new and cheaper paths may become available for the commodities in $\bar{K}$. Therefore, in Phase 2, for each commodity $k \in \bar{K}$ sorted by the highest to lowest values of $r_{k,t,*} = [c(p_{k,t-1}) - c(p^*_k)]_+$, we try to push the commodity in the shortest path $p^*_{k,t}$ connecting the origin to the destination of k where we only consider arcs with enough residual capacity left to accommodate k at period t, always updating the residual capacities of the arcs throughout the network and recording the difference $r_{k,t} = [c(p_{k,t-1}) - c(p^*_{k,t})]_+$.

Let $s_{a,t}$ and s_t, be, respectively, the sum of the values of $r_{a,k,t}$ and $r_{k,t}$ only considering the commodities that were successfully pushed in the cheaper paths in Phase 1 and Phase 2. Therefore, an upper bound on the optimal solution cost for the multi-commodity flow problem on the intermediate design at period t is given by:

$$\bar{z}(x^t) = \bar{z}(x^{t-1}) - s_{a,t} - s_t$$

We repeat Phase 1 and Phase 2 until all transition periods have been processed, where, at the beginning of each period, a single arc $a \in \hat{A}$ is selected for expansion. The upper bound on the remaining transition costs at a node i is then defined as:

$$ub(i) = \sum_{t=D(i)}^{T-1} \bar{z}(x^t)$$

The step-by-step of this procedure is outlined in Algorithm 4. Although the shortest paths $p^*_{k,a}$ and p^*_k can be pre-computed, in the worst case one additional shortest path is computed for every commodity k and period t using Bellman-Ford's algorithm in $\mathcal{O}(|N|\cdot|A|)$ steps. Therefore, the algorithm runs in $\mathcal{O}(T\cdot|K|\cdot|N|\cdot|A|)$ steps. Observe that UB-H is also a greedy heuristic; arcs are selected for expansion based on a greedy policy and commodities are pushed in new paths starting from the ones that could benefit the most from being re-routed in the network. The algorithm assures that the solution obtained for the multi-commodity flow problem in each transition period is feasible, given that the commodity paths either remain the same or are replaced by cheaper paths, always respecting the arc capacity constraints. In the worst case, when both the sequence of expansions and the order of commodities processed in each period are poorly chosen, then the total transition costs will get arbitrarily close to the worst case bound of $(T-1)z(x^0)$.

5.1.2 Lower bounds

We obtain a lower bound, by (1) assuming a best-case scenario, i.e., in each of the remaining transition periods the multi-commodity flow cost of the target design is achieved, and (2) computing the linear relaxation of INCMCF only considering the remaining transition periods.

In the linear relaxation of INCMCF, we relax the integrality constraints of the decision variables $x^t_{e,k}$ and y^t_e and only consider the $T - D(i) - 2$ remaining transition periods. The initial network design is set to be the current network design associated

Algorithm 4 Upper bound heuristic (UB-H)

input: $\bar{c}$ - flow costs on the current intermediate design
$\hat{A}$ - set of remaining candidate arcs for expansion
output: ub - upper bound on the transition costs

$ub \leftarrow 0$
$z \leftarrow \bar{c}$
$t \leftarrow 1$
while $|\hat{A}| > 1$ **do**
  $a \leftarrow$ choose arc for expansion based on the largest immediate flow cost reduction
  $\hat{A} \leftarrow \hat{A} \setminus \{a\}$
  $u_a \leftarrow u_a + w_a$
  for each commodity $k \in K$ **do** # Phase 1
    $r_{k,a,t} \leftarrow [c(p_{k,t-1}) - c(p^*_{k,a})]_+$
    $r_{k,t,*} \leftarrow [c(p_{k,t-1}) - c(p^*_k)]_+$
  end for
  $\bar{K} \leftarrow \{\}$
  $s_{a,t} \leftarrow 0$
  for each commodity $k \in K$, sorted by $r_{k,a,t}$, from highest to lowest, **do**
    if k fits in all arcs of $p^*_{k,a}$ **then**
      push k in $p^*_{k,a}$ and update residual capacities of the arcs in $p_{k,t}$
      $s_{a,t} \leftarrow s_{a,t} + r_{k,a,t}$
    else
      $\bar{K} \leftarrow \bar{K} \cup \{k\}$
    end if
  end for
  $s_t \leftarrow 0$
  for each commodity $k \in \bar{K}$, sorted by $r_{k,t,*}$ from highest to lowest **do** # Phase 2
    if k fits in all arcs of $p^*_{k,t}$ **then**
      $r_{k,t} \leftarrow [c(p_{k,t-1}) - c(p^*_{k,t})]_+$
      push k in $p^*_{k,t}$ and update residual capacities of the arcs in $p_{k,t-1}$
      $s_t \leftarrow s_t + r_{k,t}$
    end if
  end for
  $z \leftarrow z - s_{a,t} - s_t$
  $ub \leftarrow ub + z$
  $t \leftarrow t + 1$
end while
return ub

to the node i and the set of candidate arcs of expansion is limited to the arcs not yet selected for expansion by period $t = D(i)$. That is, we solve smaller linear programs adjusting the variables and constraints accordingly.

Let $z(x^{[D(i)+1,T-1]})^{LP}$ be the optimal solution cost of the linear relaxation of INCMCF for the remaining transition periods, starting from the current intermediate design $x^{D(i)}$. From Proposition 1, we saw that a valid lower bound on the transition costs for the $(T-1)$ transition periods is $(T-1) \cdot z(x^T)$. Therefore, given that the integrality gap of the linear relaxation of INCMCF may be large, we set the lower bound $lb(i)$ as:

$$lb(i) = \max\{z(x^{[D(i)+1,T-1]})^{LP}, (T - D(i) - 2) \cdot z(x^T)\}$$

that is, we take the maximum between the lower bound obtained by solving the linear relaxation of INCMCF and the transition costs assuming the best case scenario where we obtain the cheapest possible flow costs in all subsequent periods.

6 Computational Study

6.1 Instances

The proposed algorithms were used to solve a set of randomly generated instances representing different initial and target network designs.

For a given number of nodes $n = |N|$, we start by randomly sampling the x and y coordinates of the nodes in N on a grid of size $4n \times 4n$. Arcs between nodes are only created if the euclidean distance between nodes is within a given threshold of the largest euclidean distance between any pair of nodes. The capacity of an arc is chosen uniform randomly from $[Q, 2Q]$. The number of commodities is chosen uniform randomly from $[\frac{n(n-1)}{2}, n(n-1)]$. The origin and the destination nodes of each commodity k are chosen randomly and the quantity q_k associated with the commodity is chosen uniform randomly from $[1, Q]$.

For each instance, we obtain the target network design by solving the integer programming model below, where $x_{a,k}$ and y_a are integer decision variables representing, respectively, sending commodity k through arc a ($x_{a,k} = 1$) or not ($x_{a,k} = 0$) and the size of the capacity expansion of arc a:

$$\text{(TND)} \quad \min \sum_{a \in A} \bar{c}_a \cdot y_a$$

$$s.t. \quad \sum_{a \in \delta^+(i)} x_{a,k} - \sum_{a \in \delta^-(i)} x_{a,k} = \begin{cases} 1, \, i = o_k \\ -1, \, i = d_k \\ 0, \text{ o.w} \end{cases} \quad \forall i \in N, k \in K \tag{12}$$

$$\sum_{k \in \mathcal{K}} q_k \cdot x_{a,k} \leq u_a + y_a, \, \forall a \in A \tag{13}$$

$$\sum_{a \in A} \sum_{k \in K} (q_k \cdot c_a) \cdot x_{a,k} \leq \alpha \cdot z(x^0) \tag{14}$$

$$y_a \leq 2Q \cdot w_a, \forall\, a \in A \tag{15}$$

$$\sum_{a \in A} z_a \leq n \tag{16}$$

$$x_{a,k} \in \{0,1\}, \forall\, a \in A, k \in K \tag{17}$$

$$y_a \in \mathbb{Z}_+, \forall\, a \in A \tag{18}$$

$$z_a \in \{0,1\}, \forall\, a \in A \tag{19}$$

The per-unit cost of using an arc, c_a, is set to be a fraction of its length (the euclidean distance between its tail and its head) and the per-unit cost of expanding the capacity of an arc, $\bar{c}_a$, is set to ρc_a with $\rho > 1$. Constraints (12) and (13) are the typical multi-commodity flow constraints, where now the capacity of each arc a can be expanded by y_a units. Constraint (14) restricts that the multi-commodity flow costs in the target design resulting from the arc capacity expansions must be at most a fraction $\alpha \in (0,1)$ from the flow costs in the initial network design. Finally, constraints (15) limit the maximum increase in capacity for each arc to $2Q$ units and constraint (16) limits the number of arcs that can have increased capacities.

Table 1 summarizes the characteristics of the thirteen instances generated. We report: the number of nodes ($|\boldsymbol{N}|$), the number of arcs ($|\boldsymbol{A}|$), the number of expanded arcs in the target design ($|\hat{\boldsymbol{A}}|$), the number of commodities ($|\boldsymbol{K}|$), the average direct distance between connected nodes ($\boldsymbol{Avg_D}$), the longest direct distance between connected nodes ($\boldsymbol{L_D}$), the cost of the initial design ($\boldsymbol{z(x^0)}$), the cost of the target design ($\boldsymbol{z(x^T)}$) and the time it takes to solve the unsplittable multi-commodity flow problem to obtain the initial design, in seconds ($\boldsymbol{TT_{z(x^0)}}$).

Table 1: Instances generated for the incremental network design problem with multi-commodity flows

Instance	$\boldsymbol{\lvert N\rvert}$	$\boldsymbol{\lvert A\rvert}$	$\boldsymbol{\lvert\hat{A}\rvert}$	$\boldsymbol{\lvert K\rvert}$	$\boldsymbol{Avg_D}$	$\boldsymbol{L_D}$	$\boldsymbol{z(x^0)}$	$\boldsymbol{z(x^T)}$	$\boldsymbol{TT_{z(x^0)}}$
S1	8	57	6	44	5.93	10.05	68.17	61.35	0.07
S2	10	65	9	51	8.09	13.00	165.80	155.64	0.14
S3	12	125	6	79	11.12	19.31	290.21	261.13	0.38
S4	12	99	17	124	10.45	17.26	468.35	448.38	0.72
M1	16	219	10	183	12.61	23.02	769.11	692.19	3.12
M2	16	223	22	227	15.77	28.32	1,027.26	924.47	4.12
M3	16	233	18	203	17.02	28.32	930.68	837.25	5.75
M4	16	229	22	216	18.38	32.02	1,271.06	1,143.96	5.14
L1	16	183	35	208	12.48	20.62	999.05	948.97	8.37
L2	16	209	35	229	15.26	29.21	1,201.41	1,087.00	2.97
L3	20	341	17	370	14.92	27.78	1,736.65	1,566.17	109.41
L4	20	283	55	321	16.82	28.46	2,133.74	2,035.60	179.27
L5	32	893	58	384	33.21	57.69	7,507.12	7,132.49	120.42

6.2 Analysis

The goal of our computational study is twofold. First, we want to compare the performance of the proposed greedy heuristics, namely GreedySizeExp-HL, GreedySizeExp-LH, and GreedyCostRed. Second, we want to compare the performance of DF-SPE and solving the integer programming formulation using a commercial solver. The algorithms were coded in C++ and use IBM CPLEX Optimizer 12.7 to solve integer programs. All experiments were conducted in a single thread of a dedicated Intel Xeon ES-2630 2.3GHz with 250GB RAM, running Red Hat Enterprise Linux Server 7.6.

Table 2 reports the following statistics for the solutions produced by the greedy heuristics: the cost of the first intermediate design ($\boldsymbol{z(x^1)}$), the cost of the last intermediate design ($\boldsymbol{z(x^{T-1})}$), the average cost per period ($\overline{\boldsymbol{z(x)}}$), the average running time per period ($\overline{\boldsymbol{TT}}$), and the total running time of the algorithm ($\boldsymbol{TT}$). We observe that GreedySizeExp-HL outperforms GreedySizeExp-LH for all instances, which confirms our intuition that selecting arcs for expansion based on size of expansion, from highest to lowest, produces better solutions than selecting arcs in the opposite order. However, we observe too that choosing arcs for expansion based on the potential immediate flow cost reduction, as in GreedyCostRed, results in even better solutions (with the best performance in ten out of the thirteen instances). We note that the flow costs in the first intermediate designs in the best solutions found are all equal or lower than the ones in the solutions with higher cost. It is also interesting to observe how the sequence of arc capacity expansions affects the running time of the algorithms. In instances M4, L3 and L4, for example, the time required for evaluating the complete sequence of expansions determined by GreedySizeExp-LH takes, on average, 2.65 times longer than evaluating the sequence of expansions determined by GreedyCostRed. This is due to differences in the unsplittable multi-commodity flow problems that need to be solved for different sequences of expansions.

Next, we assess the performance of the exact methods. In Table 3, we report following statistics for the solutions produced by CPLEX: the average cost per period of the solution to the linear relaxation of INCMCF ($\overline{\boldsymbol{z(x)}}^{\boldsymbol{LP}}$), the integrality gap (i.e., ${}^{z(x)^{OPT} - z(x)^{LP}}/_{z(x)^{LP}}$), as a percentage ($\boldsymbol{G_{INT}}(\%)$), the number of nodes explored in the search tree ($\boldsymbol{\#Nodes}$), the number of integer solutions found ($\boldsymbol{\#Sols}$), the average cost per period of the first integer solution found ($\overline{\boldsymbol{z(x)}}^{\boldsymbol{1}}$), the average cost per period over all integer solutions found ($\boldsymbol{Avg}\ \overline{\boldsymbol{z(x)}}$), the average cost per period of the best integer solution found ($\overline{\boldsymbol{z(x)}}^{\boldsymbol{B}}$), the optimality gap between the best bound $z(x)^{BB}$ and the best integer solution found as reported by CPLEX (i.e., ${}^{z(x)^{B} - z(x)^{BB}}/_{z(x)^{BB}}$), as a percentage ($\boldsymbol{G_{OPT}(\%)}$), and the total time of the algorithm ($\boldsymbol{TT}$), where we set a time limit of 24 hours (86,400 seconds).

We observe that CPLEX was not able to find the optimal solutions for four out of the five large instances (L1, L2, L4 and L5) within the time limit of 24 hours. In fact, no integer feasible solutions were found for three out of these four instances (L1, L2 and L4), and a single integer solution was found for instance L5. We also observe that the integrality gaps (when available) are generally small, 2.70%, on average.

Table 2: Computational results for the greedy heuristics. The best solutions are highlighted in bold.

Instance	Algorithm	$z(x^1)$	$z(x^{T-1})$	$\overline{z(x)}$	$\overline{TT}$	TT
S1	GreedySizeExp-LH	68.17	64.22	66.67	0.05	0.25
	GreedySizeExp-HL	66.68	61.55	64.21	0.05	0.26
	GreedyCostRed	66.68	61.55	**63.34**	0.05	0.27
S2	GreedySizeExp-LH	165.55	158.47	162.03	0.07	0.57
	GreedySizeExp-HL	161.35	155.89	**158.87**	0.07	0.52
	GreedyCostRed	161.35	155.89	**158.87**	0.07	0.54
S3	GreedySizeExp-LH	284.23	271.85	279.43	0.22	1.09
	GreedySizeExp-HL	278.51	265.01	**272.27**	0.20	0.99
	GreedyCostRed	278.51	265.01	**272.27**	0.21	1.05
S4	GreedySizeExp-LH	468.35	451.72	462.50	0.39	6.55
	GreedySizeExp-HL	466.21	448.98	455.84	0.52	8.77
	GreedyCostRed	466.21	448.98	**455.63**	0.47	8.03
M1	GreedySizeExp-LH	766.21	708.40	742.43	2.03	18.31
	GreedySizeExp-HL	751.23	695.02	717.33	1.55	13.94
	GreedyCostRed	751.23	695.02	**717.23**	1.90	17.08
M2	GreedySizeExp-LH	1,027.26	937.63	986.35	3.26	68.41
	GreedySizeExp-HL	1,015.37	924.90	967.88	3.47	72.77
	GreedyCostRed	1,022.84	926.61	**976.00**	3.65	76.71
M3	GreedySizeExp-LH	930.67	856.61	906.55	2.83	48.04
	GreedySizeExp-HL	913.53	839.27	868.24	2.30	39.09
	GreedyCostRed	913.53	841.25	**865.98**	1.82	30.99
M4	GreedySizeExp-LH	1,271.06	1,189.10	1,250.69	3.81	80.11
	GreedySizeExp-HL	1,227.00	1,144.54	**1,161.17**	1.70	35.6
	GreedyCostRed	1,227.00	1,144.54	1,165.11	1.95	40.97
L1	GreedySizeExp-LH	999.06	956.26	983.75	3.88	131.81
	GreedySizeExp-HL	993.91	949.52	964.46	3.19	108.46
	GreedyCostRed	993.91	950.90	**963.11**	3.17	107.69
L2	GreedySizeExp-LH	1,200.64	1,090.16	1,156.62	2.28	77.5
	GreedySizeExp-HL	1,196.83	1,088.05	1,119.25	2.32	78.83
	GreedyCostRed	1,192.39	1,096.41	**1,115.92**	2.06	70.01
L3	GreedySizeExp-LH	1,731.99	1,597.81	1,685.09	18.65	298.45
	GreedySizeExp-HL	1,696.38	1,571.54	1,604.68	7.82	125.16
	GreedyCostRed	1,696.38	1,567.74	**1,601.57**	7.07	113.08
L4	GreedySizeExp-LH	2,133.74	2,037.12	2,101.08	39.50	2,133.10
	GreedySizeExp-HL	2,129.30	2,036.63	2,065.49	22.36	1,207.39
	GreedyCostRed	2,126.44	2,036.27	**2,059.09**	11.47	619.22
L5	GreedySizeExp-LH	7,507.12	7,188.57	7,417.02	16.94	965.37
	GreedySizeExp-HL	7,455.29	7,143.49	**7,225.52**	9.70	552.88
	GreedyCostRed	7,455.29	7,143.49	7,232.27	10.68	608.88

Table 3: Computational results when solving the integer programming formulation using CPLEX.

Instance	$\overline{z(x)}^{LP}$	$G_{INT}(\%)$	$\#Nodes$	$\#Sols$	$\overline{z(x)}^{1}$	$Avg\ \overline{z(x)}$	$\overline{z(x)}^{B}$	$G_{OPT}(\%)$	TT
S1	61.92	2.29	3	1	63.34	63.34	63.34	0.00	0.46
S2	156.57	0.90	3	1	157.98	157.98	157.98	0.00	0.86
S3	250.58	7.50	1	1	269.39	269.39	269.39	0.00	0.94
S4	423.09	0.95	230	5	427.26	427.18	427.13	0.00	63.8
M1	690.84	3.48	1,412	8	718.91	716.73	715.78	0.00	179.53
M2	923.49	2.96	322,791	18	952.13	951.87	951.74	0.00	82,017.62
M3	835.26	3.21	4,282	7	880.20	867.09	862.11	0.00	979.96
M4	1,143.98	1.35	857	4	1,161.28	1,160.32	1,159.43	0.00	245.17
L1	952.02	-	113,963	0	-	-	-	-	86,400.00
L2	959.39	-	168,321	0	-	-	-	-	86,400.00
L3	1,566.01	2.13	31,678	9	1,599.96	1,599.63	1,599.50	0.00	13,616.60
L4	2,054.56	-	17,162	0	-	-	-	-	86,400.00
L5	7,151.46	-	51,719	1	7,316.06	7,316.06	7,316.06	1.77	86,400.00

For the instances where more than one integer solution was found, we see that the difference between the cost of the first solution found and the cost of the last solution found is very small, 0.4% on average, which is true in part because of the small integrality gaps. We also observe that the relationship between the running times and the size of the instances in terms of the number expanded arcs is not that clear. For example, instance M2 has roughly the same number of nodes, arcs, expanded arcs and commodities as instance M4, but CPLEX takes roughly 334× longer to find its optimal solution than for M4. Instance L3, on the other hand, is larger than M2 in all aspects, but CPLEX finds its optimal solution roughly 6 times faster.

The results for DFSPE are summarized in Table 4. We report: the percentage of nodes (partial sequences) explored, out of the $2^{|\hat{A}|} - 2$ nodes in the tree ($\boldsymbol{\#PS(\%)}$), the number of complete sequences explored, i.e., the number of solutions found ($\boldsymbol{\#Sols}$), the percentage of time spent solving LPs ($\boldsymbol{TT_{LP}(\%)}$), the percentage of time spent computing upper bounds ($\boldsymbol{TT_{UB}(\%)}$), the percentage of time spent solving IPs ($\boldsymbol{TT_{IP}(\%)}$), the average cost per period of the first complete sequence explored ($\boldsymbol{\overline{z(x)}^{1}}$), the average cost per period over all complete sequences explored ($\boldsymbol{Avg\ \overline{z(x)}}$), the average cost per period of the best solution found ($\boldsymbol{\overline{z(x)}^{B}}$), and the total running time of the algorithm ($\boldsymbol{TT}$), where we again set a time limit of 24 hours (86,400 seconds).

The first thing to notice is that the algorithm takes considerably more time for finding the optimal solutions for the small and medium size instances than CPLEX. Recall that DFSPE explores partial sequences of expansions, where, in each iteration, the algorithm selects a single arc for expansion based on a pre-defined order and computes the unsplittable multi-commodity flow problem in the current intermediate design, as well as lower and upper bounds on the remaining transition costs. Therefore, the first complete sequence of expansions is only explored after solving $T - 1$ integer and linear programs, which directly affects the running times especially for the small instances. Looking at the total percentage of partial sequences explored for the instances where an optimal solution was found in under 24 hours, we see that the lower and upper bounds computed at each node manage to eliminate big portions of

Table 4: Computational results for DFSPE.

Instance	$\#PS(\%)$	$\#Sols$	$TT_{LB}(\%)$	$TT_{UB}(\%)$	$TT_{IP}(\%)$	$\overline{z(x)}^1$	$Avg\ \overline{z(x)}$	$\overline{z(x)}^B$	TT
S1	72.59	1	13.61	1.48	84.91	63.34	63.34	63.34	1.34
S2	36.67	93	25.59	1.78	72.63	158.58	158.14	157.98	16.59
S3	80.64	29	16.71	1.26	82.04	272.27	270.61	269.39	11.3
S4	28.21	1219	40.37	0.77	58.86	428.84	427.53	427.13	817.6
M1	71.03	179	25.10	2.92	71.98	717.23	716.11	715.78	1,309.14
M2	0.97	6,663	39.12	5.54	55.34	974.85	968.10	951.74	86,400.00
M3	5.00	5,973	37.67	8.96	53.37	865.98	864.03	862.11	86,400.00
M4	0.67	1,265	46.99	8.29	44.72	1,165.08	1,162.21	1,160.82	86,400.00
L1	<0.01	3,277	38.05	5.60	56.34	962.86	962.36	962.20	86,400.00
L2	<0.01	1,671	59.39	2.63	37.97	1,115.27	1,110.96	1,110.16	86,400.00
L3	6.77	91	39.08	1.26	59.66	1,601.58	1,600.38	1,599.50	86,400.00
L4	<0.01	717	64.23	2.21	33.56	2,059.09	2,058.87	2,058.70	86,400.00
L5	<0.01	591	56.04	7.39	36.58	7,231.98	7,230.75	7,223.08	86,400.00

the solution tree. For instances S2 and S4, for example, around 63% and 72% of the nodes in the solution tree were pruned, respectively. We also observe that the algorithm takes a considerable amount of time solving linear programs when computing lower bounds (38% on average, with up to 64.23% for instance L4). On the other hand, the percentage of time spent computing upper bounds is small (less than 10%). Even though the algorithm finds a large number of feasible solutions (especially for the large instances), we observe that, for all instances, the gaps between the costs of the first solutions found and of the average solutions are small (0.23%, on average). This suggests that there are likely many solutions (unsplittable flows) that either have the same cost or have only a small cost difference.

Next, we analyse the quality of the upper and lower bounds computed by DFSPE during the exploration of the partial sequences that lead to the optimal solution z^* for instance M1, which is representative of what happens for the other instances. We report: the number of remaining arcs for expansion at period t ($|\hat{\boldsymbol{A}}^{\boldsymbol{t}}|$), the cost of the unsplittable multi-commodity flow solution at period t ($\boldsymbol{z^*(x^t)}$), the average cost per period of the remaining transition periods ($\overline{\boldsymbol{z^*(x^{[t+1,T-1]})}}$), the upper bound on the average cost per period in the remaining transition periods ($\overline{\boldsymbol{UB}_{\boldsymbol{[t+1,T-1]}}}$), the gap, in percentage, between the upper bound and the true costs for the remaining transition periods ($\boldsymbol{G}_{\boldsymbol{UB}_{\boldsymbol{[t+1,T-1]}}}\boldsymbol{(\%)}$), the lower bound on the average cost per period in the remaining transition periods ($\overline{\boldsymbol{LB}_{\boldsymbol{[t+1,T-1]}}}$), and the gap, in percentage, between the lower bound and the true costs for the remaining transition periods ($\boldsymbol{G}_{\boldsymbol{LB}_{\boldsymbol{[t+1,T-1]}}}\boldsymbol{(\%)}$). The results are shown in Table 5. We note that the gaps between both the upper and lower bounds and the optimal solution found ($z^* = 6,442.00$) are relatively small throughout the exploration of the partial sequences (0.42%, on average). In particular, when half of the complete sequence was explored, the values of $\boldsymbol{G}_{\boldsymbol{UB}_{\boldsymbol{[t+1,T-1]}}}\boldsymbol{(\%)}$ are all less or equal than 0.05%. The gaps for the lower bounds are, in most part, higher, but still relatively small (1.60%, on average).

Finally, in Table 6, we summarize and compare the performances of the best variant of the greedy heuristics (GreedyCostRed), the branch and bound algorithm used in CPLEX (B&B) and DFSPE, where we also report the gap, in percentage, between the best solutions found by the algorithms and the best known solution found amongst all methods ($\boldsymbol{G}_{\boldsymbol{BKS}}\boldsymbol{(\%)}$). The optimal solution for each instance is marked

Table 5: Upper and lower bounds for instance M1.

t	$\lvert\hat{A}^t\rvert$	$z^*(x^t)$	$\overline{z^*(x^{[t+1,T-1]})}$	$\overline{UB_{[t+1,T-1]}}$	$G_{UB_{[t+1,T-1]}}(\%)$	$\overline{LB_{[t+1,T-1]}}$	$G_{LB_{[t+1,T-1]}}(\%)$
0	10	769.11	715.78	723.49	1.06	690.84	3.48
1	9	751.23	711.35	719.86	1.18	692.18	2.69
2	8	738.10	707.52	717.46	1.38	692.18	2.17
3	7	725.62	704.51	710.78	0.88	692.18	1.75
4	6	717.76	701.86	702.08	0.03	692.18	1.38
5	5	710.58	699.68	699.95	0.04	692.18	1.07
6	4	704.75	697.99	698.35	0.05	692.18	0.83
7	3	701.04	696.46	696.59	0.02	692.19	0.61
8	2	697.90	695.02	695.02	0.00	692.19	0.40
9	1	695.02	-	-	-	-	-

with an asterisk.

Although the costs of the first solutions found by DFSPE are, in some cases, more expensive than the first solutions found by CPLEX, we note that, for the larger instances, DFSPE finds better solutions overall than both CPLEX and GreedyCostRed. Recall that in DFSPE, arcs are selected for expansion based on a pre-defined order following the same policy used in GreedyCostRed (i.e. by the potential largest immediate reduction in flow costs). This explains why many of the first solutions found by DFSPE are the same as the ones found by GreedyCostRed. However, the policies for breaking ties in these methods are different; GreedyCostRed break ties by selecting arcs with the highest size of expansion first, then by largest number of commodity paths using the arc and finally by randomly selecting an arc. DFSPE, on the other hand, break ties by selecting the arc that yields the lowest upper bound on the remaining transition costs. Therefore, we see that some of the first solutions found by DFSPE are better than the ones found by GreedyCostRed. The results also show that the best variant of the greedy heuristics was only able to find the optimal solution for L1, which is the smallest instance in the set. The average gap between the solutions found by GreedyCostRed and the optimal solutions found for the small and medium size instances is of only 0.6%, which is relatively small. Overall, DFPSE outperformed both CPLEX in all but one of the larger instances, both in the quality of the best solutions found and in the number of feasible solutions found.

7 Final Remarks

We have introduced a new incremental network design problem motivated by the expansion of hub capacities in package express service networks, where the cost of each period is given by the solution to an unsplittable multi-commodity flow problem. This is the first time in the literature where the period problem in an incremental network design problem is NP-hard. We proposed an integer programming formulation for its solution, as well as a set of greedy heuristics and an exact method that explores partial sequences of arc capacity expansions in a depth-first manner. We

Table 6: Computational results for the various algorithms. The best first and overall solutions found are highlighted in bold.

Instance	Algorithm	$\#Sols$	$\overline{z(x)}^1$	$Avg\ \overline{z(x)}$	$\overline{z(x)}^B$	$G_{BKS}(\%)$	TT
S1	GreedyCostRed	-	**63.34***	-	**63.34***	0.00	0.25
	B&B	1	**63.34***	63.34	**63.34***	0.00	0.46
	DFSPE	1	**63.34***	63.34	**63.34***	0.00	1.34
S2	GreedyCostRed	-	158.87	-	158.87	0.56	0.54
	B&B	1	**157.98***	157.98	**157.98***	0.00	0.86
	DFSPE	93	158.58	158.14	**157.98***	0.00	16.59
S3	GreedyCostRed	-	272.27	-	272.27	1.06	1.05
	B&B	1	**269.39***	269.39	**269.39***	0.00	0.94
	DFSPE	29	272.27	270.61	**269.39***	0.00	11.30
S4	GreedyCostRed	-	428.84	-	428.84	0.40	8.03
	B&B	5	**427.26**	427.18	**427.13***	0.00	63.80
	DFSPE	1219	428.84	427.53	**427.13***	0.00	817.60
M1	GreedyCostRed	-	717.23	-	717.23	0.20	17.08
	B&B	8	718.91	716.73	**715.78***	0.00	179.53
	DFSPE	179	**717.23**	716.11	**715.78***	0.00	1,309.14
M2	GreedyCostRed	-	976.00	-	976.00	2.47	76.71
	B&B	18.00	**952.13**	951.87	**951.74***	0.00	82,017.62
	DFSPE	6,663	974.85	968.10	**951.74***	0.00	86,400.00
M3	GreedyCostRed	-	**865.98**	-	865.98	0.45	30.99
	B&B	7	880.20	867.09	**862.11***	0.00	979.96
	DFSPE	5973	**865.98**	864.03	**862.11***	0.00	86,400.00
M4	GreedyCostRed	-	1,165.11	-	1,165.11	0.37	40.97
	B&B	4	**1,161.28**	1,160.32	**1,159.43***	0.00	245.17
	DFSPE	1,265	1,165.08	1,162.21	1,160.82	0.12	86,400.00
L1	GreedyCostRed	-	963.11	-	963.11	0.10	107.69
	B&B	0	-	-	-	-	86,400.00
	DFSPE	3277	**962.86**	962.36	**962.20**	0.00	86,400.00
L2	GreedyCostRed	-	1,115.92	-	1,115.92	0.52	70.01
	B&B	0	-	-	-	-	86,400.00
	DFSPE	1,671	**1,115.27**	1,110.96	**1,110.16**	0.00	86,400.00
L3	GreedyCostRed	-	1,601.57	-	1,601.57	0.13	113.08
	B&B	9	**1,599.96**	1,599.63	**1,599.50***	0.00	13,616.6
	DFSPE	91	1,601.57	1,600.38	**1,599.50***	0.00	86,400.00
L4	GreedyCostRed	-	2,059.09	-	2,059.09	0.02	619.22
	B&B	0	-	-	-	-	86,400.00
	DFSPE	717	**2,059.09**	2,058.87	**2,058.70**	0.00	86,400.00
L5	GreedyCostRed	-	7,232.27	-	7,232.27	0.13	608.88
	B&B	1	7,316.06	7,316.06	7,316.06	1.27	86,400.00
	DFSPE	591	**7,231.98**	7,230.75	**7,223.08**	0.00	86,400.00

provide worst-case analyses for the proposed greedy heuristics and compared the efficiency of the algorithms on a set of artificial instances against traditional branch and bound methods used in commercial solvers. The results showed that the partial enumeration heuristic outperforms commercial solvers for large instances, where no feasible solutions could be found by the branch and bound algorithm under a time limit of 24 hours.

A variant of the incremental network design problem with multi-commodity flows, in which it is possible to temporarily expand the capacity of arcs, i.e., expand the capacity of arcs that are not are not expanded in the target design for part of the transition period, will be tackled in a future research. We show that using temporary arc expansions, even if it means reaching the target design in more periods (i.e., lengthening the transition period), can result in lower transition costs.

More practical aspects and constraints need to be incorporated in the model presented in this chapter to be able to develop effective real-world decision support for package express carriers. A more thorough investigation of temporary capacity expansions may also be needed and beneficial before developing practical decision support tools to better understand the trade-offs between operating profits and the timing and size of investments in additional capacity.

References

J. Andre, F. Bonnans, and L. Cornibert. Optimization of capacity expansion planning for gas transportation networks. European Journal of Operational Research, 197(3): 1019–1027, 2009.

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

B. Cavdaroglu, E. Hammel, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace. Integrating restoration and scheduling decisions for disrupted interdependent infrastructure systems. Annals of Operations Research, 203(1):279–294, 2013.

M. Çelik. Network restoration and recovery in humanitarian operations: Framework, literature review, and research directions. Surveys in Operations Research and Management Science, 21(2):47–61, 2016.

R. DeBlasio and C. Tom. Standards for the smart grid. In 2008 IEEE Energy 2030 Conference, pages 1–7. IEEE, 2008.

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

H. Farhangi. The path of the smart grid. IEEE power and energy magazine, 8(1):18–28, 2009.

M. Gendreau, J.-Y. Potvin, A. Smires, and P. Soriano. Multi-period capacity expansion for a local access telecommunications network. European Journal of Operational Research, 172(3):1051–1066, 2006.

N.-S. Hsu, W.-C. Cheng, W.-M. Cheng, C.-C. Wei, and W. W.-G. Yeh. Optimization and

capacity expansion of a water distribution system. Advances in Water Resources, 31 (5):776–786, 2008.

C. Jablonowski, H. Ramachandran, L. Lasdon, et al. Modeling facility-expansion options under uncertainty. SPE Projects, Facilities & Construction, 6(04):239–247, 2011.

J. Jeong and D. Culler. Incremental network programming for wireless sensors. In 2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004., pages 25–33. IEEE, 2004.

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

H. Luss. Operations research and capacity expansion problems: A survey. Operations research, 30(5):907–947, 1982.

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

T. V. Mathew and S. Sharma. Capacity expansion problem for large urban transportation networks. Journal of Transportation Engineering, 135(7):406–415, 2009.

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

S. G. Nurre, B. Cavdaroglu, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace. Restoring infrastructure systems: An integrated network design and scheduling (inds) problem. European Journal of Operational Research, 223(3):794–806, 2012.

K.-J. Wang and S.-H. Lin. Capacity expansion and allocation for a semiconductor testing facility under constrained budget. Production Planning & Control, 13(5):429–437, 2002.