CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Exact Algorithms for the Multicommodity Uncapacitated Fixed-Charge Network Design Problem

Carlos Armando Zetina
Ivan Contreras
Jean-François Cordeau

November 2017

CIRRELT-2017-69

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Exact Algorithms for the Multicommodity Uncapacitated Fixed-Charge Network Design Problem

Carlos Armando Zetina[1,2,*], Ivan Contreras[1,2], Jean-François Cordeau[1,3]

1. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
2. Department of Mechanical and Industrial Engineering, Concordia University, 1515 St-Catherine Street W., EV4 139, Montréal, Canada H3G 1M8
3. Department of Logistics and Operations Management, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Abstract. We present two exact algorithms based on Benders decomposition for solving the multicommodity uncapacitated fixed-charge network design problem. The first implements Benders decomposition within a branch-and-cut framework. We present implementation techniques crucial for the algorithm's efficiency such as cut selection criteria and core point definition and highlight their impact on the solution time. Our second algorithm combines a cut-and-solve strategy with Benders decomposition. We also describe a slope scaling metaheuristic to efficiently obtain high quality solutions. We perform extensive computational experiments on benchmark instances to compare the efficiency of the proposed algorithms. Both exact algorithms provide a speedup between one and two orders of magnitude compared to a state-of-the-art general-purpose MIP solver.

Keywords. Network design, Benders decomposition, lift-and-project cuts, cut-and-solve, slope scaling.

Acknowledgements. This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) (Grants 418609-2012 and 04959-2014). This support is gratefully acknowledged.

---

* Corresponding author: c_zetina@encs.concordia.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2017

1. Introduction

Network design problems (NDPs) lie at the heart of network planning in telecommunications and transportation systems, namely in airline, rail and motor carrier industries. These problems involve arc activation and routing decisions to ensure cost-effective paths between origin-destination (OD) pairs to satisfy demand flows. They can be classified into *single* and *multicommodity* variants. In single-commodity models, demand is defined at the nodes of a graph, which are partitioned into origin, destination and transshipment nodes. In the case of multicommodity models, demand relates to OD node pairs. The applicability of NDPs spans several domains ranging from transportation and logistics to biology and statistics. Highway design, electrical routing and supply chain management are a few common applications. Other less obvious uses are in DNA sequencing and matrix rounding equivalence (Ahuja et al. 1995). A large body of literature exists on the application of these network models in problems such as personnel scheduling (Bartholdi et al. 1980, Balakrishnan and Wong 1990), service network design (Crainic 2000, Andersen et al. 2009, Crainic and Rousseau 1986), and logistics network design (Geoffrion and Graves 1974, Santoso et al. 2005, Cordeau et al. 2006). These and others such as those mentioned in Ahuja et al. (1991, 1993) demonstrate the impact of NDPs in our daily lives.

In this paper we focus on a fundamental NDP referred to as the *multicommodity uncapacitated fixed-charge network design problem* (MUFND). The problem is defined on a directed graph and considers a set of commodities each with an origin, a destination and a demand quantity. The objective is to select a subset of arcs to install in order to route all commodities from their origins to their destinations at minimal cost. The MUFND is $\mathcal{NP}$-hard (Johnson et al. 1978) and generalizes a large class of well-known problems such as the *traveling salesman*, *uncapacitated lot-sizing*, and *Steiner network design problems* (Ortega and Wolsey 2003). Other variants of the problem have been studied and they differ by the underlying structure of the commodities: single or multiple sources and single or multiple destinations.

From a methodological point of view, there exists an extensive collection of solution algorithms for different NDPs. Some authors have approached these problems from a polyhedral perspective by proposing families of valid inequalities that strengthen well-known formulations (Bienstock and Günlük 1996, Günlük 1999, Atamtürk 2002, Atamtürk and Rajan 2002, Raack et al. 2011). Others have focused on the development of decomposition methods for solving them (Cruz et al. 1998, Randazzo and Luna 2001, Crainic et al. 2001, Frangioni and Gendron 2009, 2013, Frangioni and Gorgone 2014), while others have proposed heuristic algorithms that obtain high quality solutions (Balakrishnan et al. 1989, Kim and Pardalos 1999, Ghamlouche et al. 2003, 2004, Crainic et al. 2004, Katayama et al. 2009, Yaghini et al. 2015).

In the case of MUFND, the first proposed solution algorithm is an add-drop heuristic by Billheimer and Gray (1973). Other heuristics are those of Dionne and Florian (1979), Boffey and Hinxman (1979), Los and Lardinois (1982), Kratica et al. (2002). Lamar et al. (1990) propose a novel form of iteratively obtaining strengthened dual bounds from a weaker formulation by adjusting capacity constraints. Balakrishnan et al. (1989) propose

a dual ascent algorithm to obtain solutions to large-scale instances with up to 600 arcs and 1,560 commodities. Their method obtains solutions that are between 1% and 4% away from optimality in less than 150 seconds. With respect to exact methods, Magnanti et al. (1986) propose the use of a Benders decomposition tailored for the undirected version of the MUFND. They are able to solve instances with up to 130 arcs and 58 commodities to proven optimality. Holmberg and Hellstrand (1998) propose a branch-and-bound algorithm that uses Lagrangean relaxation bounds to solve directed instances with up to 1,000 arcs and 600 commodities. Recently, Fragkos et al. (2017) used Benders decomposition to solve a multi-period extension of the MUFND. They experiment with the use of Pareto-optimal cuts and with the unified cut approach of Fischetti et al. (2010), obtaining significant computational gains with the latter on instances with up to 318 arcs, 100 commodities and 108 time periods. To the best of our knowledge, these are the current state-of-the-art exact methods for the undirected and directed MUFND.

The main contributions of this article are the following:

- We present an exact branch-and-cut algorithm based on a Benders reformulation to solve large-scale instances of the MUFND. This reformulation is obtained by projecting out a large number of continuous variables from the well-known disaggregated arc-based formulation of the MUFND.

- We present exact separation procedures to efficiently generate feasibility and optimality cuts and address crucial implementation techniques that should be considered when generating cuts at fractional and integer points such as cutting frequency, cut separation, and core point selection within a single enumeration tree.

- We provide insights on the importance that different core point selection strategies have on the overall convergence of the algorithm. We propose a straightforward but effective rule for core point selection that provides a significant speed-up for the MUFND.

- We study the relationship between feasibility cuts and cutset inequalities. In particular, we show that while all cutsets correspond to rays of the dual subproblem, they are not all extreme rays. We also show that not all feasibility cuts obtained from extreme rays are cutset inequalities and hence neither is a subset of the other.

- We introduce a new family of valid inequalities, referred to as *Benders lift-and-project cuts*, to strengthen the Benders reformulation. We assess their impact in terms of closing the optimality gap of the linear programming (LP) relaxation of the Benders reformulation.

- We develop a slope scaling metaheuristic to obtain feasible solutions for the MUFND that exploits the information generated by the Benders dual subproblems as a perturbation mechanism to diversify the search.

- We present an exact cut-and-solve algorithm in which the sparse and dense problems are efficiently solved with our Benders-based branch-and-cut algorithm.

- We perform extensive computational experiments on several sets of benchmark instances to asses the performance of our algorithms. The proposed algorithms are up to two orders of magnitude faster than the state-of-the-art MIP solver CPLEX 12.7.1 and solve instances of larger size than those previously presented.

The remainder of the paper is organized as follows. Section 2 provides a formal definition of the MUFND and presents the arc-based formulation. Section 3 describes the Benders reformulation and presents a comparative analysis between Benders feasibility cuts and cutset inequalities. Section 4 details the enhancements implemented in our branch-and-cut algorithm. In Section 5, we present our second algorithmic framework, an exact cut-and-solve algorithm. Summarized results of our extensive computational experiments are given in Section 6, while Section 7 presents concluding remarks and future lines of research.

2. Problem Definition

The MUFND is defined on a directed graph $G = (N, A)$ where N is a set of nodes, A is a set of arcs and K is a set of commodities each defined by the tuple (o_k, d_k, W_k) representing the origin, destination, and demand quantity of a commodity $k \in K$, respectively. The key feature of this problem is its use in evaluating the trade-off between infrastructure investment and operational costs. The former is modeled by the fixed cost paid for using an arc f_{ij} joining node i to node j. The latter is modeled by a linear transportation cost c_{ij}^k paid per unit of commodity k routed on arc (i, j). The goal is to route all commodities from origins to destinations at minimal cost.

There are two well-known formulations for this problem classified as the *aggregated* and *disaggregated* arc-based formulations. Both use a set of binary variables y_{ij} to model whether arc (i, j) is installed or not and a set of continuous variables x_{ij}^k to represent the fraction of commodity k's demand routed on arc (i, j). In this study, we use the disaggregated formulation since it is known that Benders decomposition performs better with tighter formulations (Magnanti and Wong 1981). The MUFND can be stated as follows:

$$\text{(P)} \quad \text{minimize} \sum_{(i,j)\in A} f_{ij} y_{ij} + \sum_{k\in K} \sum_{(i,j)\in A} W^k c_{ij}^k x_{ij}^k \tag{1}$$

$$\text{subject to} \quad \sum_{j\in N} x_{ji}^k - \sum_{j\in N} x_{ij}^k = \begin{cases} -1 & \text{if } i = o_k \\ 1 & \text{if } i = d_k \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N, \forall k \in K \tag{2}$$

$$x_{ij}^k \leq y_{ij} \quad \forall (i,j) \in A, k \in K \tag{3}$$

$$x_{ij}^k \geq 0 \quad \forall (i,j) \in A, k \in K \tag{4}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i,j) \in A. \tag{5}$$

The objective function (1) is the total cost of the network including both the installation and routing costs for all arcs and commodities. Flow conservation constraints (2)

ensure that the demand for all commodities is routed from origin to destination. Constraint set (3) assures that no flow is sent through an arc that has not been installed, while (4) and (5) are the non-negativity and integrality conditions of x and y, respectively.

Note that depending on the instance data, P is a valid formulation for the Steiner tree problem (all commodities share the same origin and no transportation costs exist) or the travelling salesman problem (all arcs have the same fixed cost and the underlying graph is complete) (Holmberg and Hellstrand 1998). This shows the wide range of special problems generalized by the MUFND and as such the inherent difficulty in developing an efficient exact algorithm for such a general model.

3. Benders Decomposition for MUFND

Benders decomposition is a well-known solution method for mixed integer programming problems (Benders 1962). It splits large formulations into two problems, an integer master problem and a linear subproblem. The principle behind Benders decomposition is the projection of a large problem into a smaller subspace, namely the space of the integer constrained variables. As a consequence, the projected model contains an exponential number of constraints known as Benders cuts, indexed by the extreme points and extreme rays of a special linear programming problem known as the dual subproblem (DSP) or slave problem. Noting that not all Benders cuts are necessary to obtain the optimal solution, Benders (1962) proposed to relax these and iteratively solve the integer master problem to obtain a lower bound on the integral optimal solution value and then substitute the solution into the dual subproblem thereby obtaining an upper bound and a Benders cut to be added to the master problem. This is to be repeated until the upper and lower bounds are within a given optimality tolerance ϵ.

In this section we present a Benders reformulation of P and provide a comparative analysis between Benders feasibility cuts and cutset inequalities. We then describe in Section 4 how this reformulation can be efficiently solved with a branch-and-cut algorithm.

3.1. Benders Reformulation

The following steps describe the process of applying Benders decomposition to formulation P of the MUFND. Note that by fixing $y = \bar{y}$, where $\bar{y} \in Y$ and $Y = \mathbb{B}^{|A|}$ denotes the set of binary vectors associated with the y_{ij} variables, we obtain a linear program in x that is easily solved. This new linear program will be denoted as the primal subproblem (PSP) and has the following form:

$$\text{(PSP)} \quad \text{minimize} \sum_{k \in K} \sum_{(i,j) \in A} W^k c_{ij}^k x_{ij}^k$$

$$\text{s.t.} \quad \sum_{j \in N} x_{ji}^k - \sum_{j \in N} x_{ij}^k = \begin{cases} -1 & \text{if } i = o_k \\ 1 & \text{if } i = d_k \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N, \forall k \in K \tag{6}$$

$$x_{ij}^k \leq \bar{y}_{ij} \qquad \forall (i,j) \in A, k \in K \tag{7}$$

$$x_{ij}^k \geq 0 \qquad \forall (i,j) \in A, k \in K.$$

Note that PSP can be split into $|K|$ subproblems PSP_k, one for each commodity. Let λ and μ denote the dual variables of constraints (6) and (7), respectively. From strong duality, each PSP_k can be substituted by its LP dual, denoted as DSP_k, of the form:

$$\begin{aligned} (\text{DSP}_k) \quad \text{maximize} \quad & \lambda^k_{d_k} - \lambda^k_{o_k} - \sum_{(i,j)\in A} \mu^k_{ij}\bar{y}_{ij} \\ \text{subject to} \quad & \lambda^k_j - \lambda^k_i - \mu^k_{ij} \leq W^k c^k_{ij} \qquad \forall (i,j) \in A \\ & \mu^k_{ij} \geq 0 \qquad \forall (i,j) \in A \\ & \lambda^k_i \in \mathbb{R} \qquad \forall i \in N. \end{aligned} \tag{8}$$

From the Farkas Lemma, we know that for a given $k \in K$, PSP_k is feasible if and only if

$$\bar{y} \in R_k = \left\{ y \in Y |\ 0 \geq \lambda^k_{d_k} - \lambda^k_{o_k} - \sum_{(i,j)\in A} \mu^k_{ij}\bar{y}_{ij},\ \forall (\lambda^k, \mu^k) \in \Theta \right\},$$

where Θ is the recession cone of DSP_k. The inequalities that define R_k are known as Benders feasibility cuts and, although by Farkas Lemma there exists an infinite number of them, only those associated with the (finite) set of extreme rays are necessary. Therefore, we use the representation of each polyhedron associated with each DSP_k in terms of its extreme points and extreme rays to determine whether PSP is infeasible or feasible and bounded. Let $\text{Ext}(\text{DSP}_k)$ and $\text{Opt}(\text{DSP}_k)$ denote the sets of extreme rays and extreme points of DSP_k, respectively.

If, for a given $y \in Y$, there exists at least one $k \in K$ and one extreme ray $(\lambda, \mu) \in \text{Ext}(\text{DSP}_k)$ for which

$$0 < \lambda^k_{d_k} - \lambda^k_{o_k} - \sum_{(i,j)\in A} \mu^k_{ij}\bar{y}_{ij},$$

then DSP_k is unbounded and PSP is infeasible. However, if

$$0 \geq \lambda^k_{d_k} - \lambda^k_{o_k} - \sum_{(i,j)\in A} \mu^k_{ij}\bar{y}_{ij},$$

for each $k \in K$ and each extreme ray $(\lambda, \mu) \in \text{Ext}(\text{DSP}_k)$, then all DSP_k are bounded and the PSP is feasible. The optimal value of each DSP_k is then equal to

$$\max_{(\lambda,\mu)\in \text{Opt}(\text{DSP}_k)} \lambda^k_{d_k} - \lambda^k_{o_k} - \sum_{(i,j)\in A} \mu^k_{ij}\bar{y}_{ij}.$$

Using continuous variables z_k for the transportation cost of each commodity $k \in K$,

the *Benders reformulation* associated with P is

$$\text{(MP)} \quad \text{minimize} \sum_{(i,j)\in A} f_{ij}y_{ij} + \sum_{k\in K} z_k \tag{9}$$

$$\text{s.t.} \qquad z_k \geq \lambda^k_{d_k} - \lambda^k_{o_k} - \sum_{(i,j)\in A} \mu^k_{ij} y_{ij} \qquad \forall k \in K, (\lambda,\mu)_k \in \text{Opt}(\text{DSP}_k) \tag{10}$$

$$0 \geq \bar{\lambda}^k_{d_k} - \bar{\lambda}^k_{o_k} - \sum_{(i,j)\in A} \bar{\mu}^k_{ij} y_{ij} \qquad \forall k \in K, (\bar{\lambda},\bar{\mu}) \in \text{Ext}(\text{DSP}_k) \tag{11}$$

$$y \in \{0,1\}^{|A|}.$$

MP exploits the decomposability of the subproblems by disaggregating the feasibility and optimality cuts per commodity. This type of *multi-cut* reformulation leads to a better approximation of the transportation costs at each iteration, which has been empirically shown to improve solution times (Magnanti et al. 1986, Contreras et al. 2011).

3.2. Benders Feasibility Cuts and Cutset Inequalities

Given $k \in K$ and $S \subset N$ such that $o_k \in S$ and $d_k \notin S$, the following cutset inequality is valid for the MUFND (see, Magnanti et al. 1986):

$$\sum_{(i,j)\in\delta(S,N\backslash S)} y_{ij} \geq 1, \tag{12}$$

where $\delta(S, N\backslash S) = \{(i,j) \in A | i \in S, j \in N\backslash S\}$. Cutset inequalities are sufficient to guarantee feasibility in the MUFND and hence can substitute the feasibility cuts (11). The advantage of using cutset inequalities is that they can be efficiently separated by solving a minimum cut problem over an auxiliary network. Algorithms such as the Edmonds-Karp algorithm (Edmonds and Karp 1972) and breadth first search are efficient in separating cutset inequalities for fractional and integer solutions, respectively.

There is also a close relationship between cutset inequalities and Benders feasibility cuts. Some of their characteristics for the capacitated case have been explored in Costa et al. (2009), where the authors show that for a given commodity $k \in K$ a cutset defined by $\delta(S, N\backslash S)$, where $o_k \in S$ $d_k \notin S$, can also be expressed as a ray (not necessarily an extreme ray) of DSP_k and therefore can be considered a Benders feasibility cut. It is easy to see that the same holds true for the uncapacitated case.

To the best of our knowledge, there has not been much effort in attempting to classify the extreme rays of DSP_k. Although one may be tempted to think that there is a one-to-one relationship between cutset inequalities and the extreme rays of DSP_k, we next show by a counterexample that this is not the case. In our example, there exist extreme rays that when transformed to Benders cuts define cutset inequalities, non-negativity constraints or another type of constraints which we call *consistency* constraints.

We begin by pointing out that the set of extreme rays is independent of the commodity $k \in K$. Although this is mathematically obvious, it seems counter-intuitive. Note that the extreme rays can be obtained as the extreme points of the recession cone of DSP_k

and an additional normalization constraint. This polytope is completely independent of $k \in K$. Its dependency is lost when taking its recession cone which homogenizes DSP_k's constraints. It is only when we define the corresponding Benders cut that we need to select $k \in K$. As a result, the set of extreme rays is the same for all $k \in K$ with only the resulting Benders feasibility cut varying.

Our example is based on the network given in Figure 1. Using the software *Polymake* (Assarf et al. 2017), we have calculated the extreme rays of the corresponding recession cone, which are listed in Table 1.

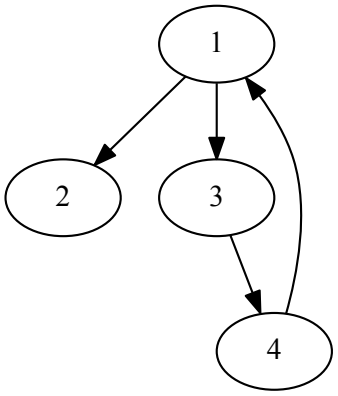

Figure 1: Sample Network

Table 1: List of Extreme Rays

	λ_1	λ_2	λ_3	λ_4	μ_{12}	μ_{13}	μ_{34}	μ_{41}	Description
Ray1	0	-1	0	0	0	0	0	0	Consistency
Ray2	0	0	0	0	0	0	0	1	Non-Negativity
Ray3	0	0	0	0	0	0	1	0	Non-Negativity
Ray4	0	0	0	0	0	1	0	0	Non-Negativity
Ray5	0	0	-1	0	0	0	1	0	Cutset(S={3})
Ray6	1	1	1	0	0	0	0	1	Cutset(S={4})
Ray7	-1	-1	0	0	0	1	0	0	Cutset(S={1,2})
Ray8	1	1	0	0	0	0	0	1	Cutset(S={3,4})
Ray9	0	0	1	0	0	1	0	0	Cutset(S={1,2,4})
Ray10	0	1	0	0	1	0	0	0	Cutset(S={1,3,4})
Ray11	-1	-1	-1	0	0	0	1	0	Cutset(S={1,2,3})

Note that Ray1 leads to one of the following when substituted into the Benders cut for a given commodity k:

- *Case 1*: Node 2 is neither the origin nor the destination of commodity k then $0 \geq 0$.
- *Case 2*: Node 2 is the destination of commodity k then $0 \geq -1$.
- *Case 3*: Node 2 is the origin of commodity k then $0 \geq 1$.

Note that Case 3 leads to a contradiction. This is because the cut obtained from the extreme ray has identified that the topology of the underlying network does not allow for node 2 to be the origin of a commodity since it has no outgoing arc. We refer to this type of cuts as *consistency* cuts and they are characterized by rays having all values of $\mu = 0$.

The 2nd, 3rd and 4th extreme rays lead to non-negativity constraints of arcs $(4, 1), (3, 4)$ and $(1, 3)$, respectively. Note that no non-negativity constraints are obtained for y_{12}.

We note that for a given extreme ray, either $\lambda_i \geq 0 \ \forall i \in N$ or $\lambda_i \leq 0 \ \forall i \in N$. We give the following interpretation for each of the remaining extreme rays:

- If $\lambda_i \geq 0 \ \forall i \in N$ then $[i \in N \backslash S \iff \lambda_i = 1]$ and $[i \in S \iff \lambda_i = 0]$,
- If $\lambda_i \leq 0 \ \forall i \in N$ then $[i \in N \backslash S \iff \lambda_i = 0]$ and $[i \in S \iff \lambda_i = -1]$.

With this interpretation, the values of $\mu_{ij} = 1$ coincide with $i \in S$ and $j \notin S$. Hence, the rest of the extreme rays are cutset inequalities for sets S shown in Table 1. Depending on the origin and destination of a commodity k, an extreme ray defines a valid cutset inequality for it.

One last thing we note is that there exist cutset inequalities that are not extreme rays of DSP. In this particular case, assuming $(o_k, d_k, W_k) = (3, 2, 1)$ and $S = \{1, 3\}$. We note that the corresponding cutset inequality would be $y_{12} + y_{34} \geq 1$. This inequality is not indexed by any extreme ray but is instead defined by the ray resulting from summing Ray3 and Ray10.

This example shows that not all cutsets are extreme rays. It also shows that not all Benders cuts obtained from extreme rays are cutset inequalities and hence neither is a subset of the other.

Preliminary tests showed that the use of cutset inequalities to ensure feasibililty outperforms the use of Benders feasibility cuts from extreme rays. We attribute this to the shorter computation time needed to separate cutsets compared to obtaining extreme rays of DSP. A more detailed comparison is presented in Appendix A of the online supplement of this paper, where we present a novel cut generating linear program guaranteed to obtain an extreme ray instead of using the rays obtained from simplex pivots.

In our procedure we first ensure feasibility of the solution before attempting to generate Benders optimality cuts. The efficiency of cutset separation leads to less computational time spent solving problems to ensure feasibility leading our procedure to start generating Pareto-optimal Benders cuts within a few seconds.

4. A Branch-and-Cut Algorithm for MUFND

Since its introduction, Benders decomposition has been used to solve several difficult problems such as airline scheduling (Cordeau et al. 2001, Papadakos 2009), facility

location (Geoffrion and Graves 1974, Fischetti et al. 2017), hub network design (Contreras et al. 2011), and fixed-charge network design (Costa 2005). Although the initially proposed algorithm suffered from slow convergence, through the years researchers have successfully devised enhancements to significantly increase its speed. Nowadays, modern implementations of Benders decomposition incorporate additional enhancements strategies such as generation of strong cuts, cut selection, stabilization, lower bound reinforcing, and solving one enumeration tree (Botton et al. 2013, Naoum-Sawaya and Elhedhli 2013, Adulyasak et al. 2015, Fischetti et al. 2017, Rahmaniani et al. 2017, Bodur and Luedtke 2017, Ortiz-Astorquiza et al. 2017). Choosing the best enhancements for a given problem is not a trivial task since each improves the performance in a different manner.

Our Benders-based branch-and-cut algorithm employs the following algorithmic features: a preprocessing routine, the generation of Pareto-optimal cuts, a core point selection criterion, lower bound strengthening via lift-and-project cuts, a slope scaling metaheuristic, and fine-tuning of cut parameters. In the following sections we explain each of the aforementioned enhancements.

4.1. Preprocessing

Since MP is a Benders reformulation of the original formulation P, by relaxing the integrality constraints and adding all Benders cuts to MP, we would obtain the LP relaxation solution of P. This is particularly important to note when implementing Benders decomposition in a single enumeration tree. One of the recent common practices is to solve MP as a linear program with a cutting plane algorithm and use the Benders cuts generated as part of the problem definition declared to the MIP solver. General-purpose solvers use this information to increase lower bounds, infer good branching rules, fix variables in their preprocessing routine and reduce the underlying linear program's size.

In our algorithm, we solve MP as a linear program before declaring it within the MIP framework. However, instead of defining the problem with all Benders cuts generated so far, we only include the Benders cuts that are binding at the optimal LP solution. This guarantees that we obtain the LP optimal value before attempting to separate Benders cuts but pass on only the essential information to the general-purpose solver and avoid declaring an excessively large problem. This step, while obvious, significantly helps the solution process.

4.2. Pareto-optimal Cut Separation

Since the seminal paper on cut selection in Benders decomposition by Magnanti and Wong (1981), Pareto-optimal cuts have become the standard practice. The approach applies to problems for which there is an infinite number of alternative optimal solutions to DSP_k and therefore Benders optimality cuts. This is particularly the case in network design problems known for their primal degeneracy. The authors define cut dominance as follows. Given two cuts defined by dual solutions u and u^1 of the form $z \geq f(u) + yg(u)$ and $z \geq f(u^1) + yg(u^1)$, respectively, the cut defined by u dominates that defined by u^1 if and only if $f(u) + yg(u) \geq f(u^1) + yg(u^1)$ with strict inequality holding for some feasible

y of MP. If a cut defined by u is not dominated by any other optimality cut, then this cut is said to be a Pareto-optimal Benders cut.

In general, to obtain Pareto-optimal Benders cuts an additional linear program must be solved at each iteration. This additional linear program is the same as the dual subproblem with two exceptions. The first is that a point y^0 in the relative interior of the master problem space, known as a core point, replaces the master problem solution $\bar{y}$ in the objective function (8). The second is that an equality constraint is added to ensure that the obtained solution belongs to the set of alternative optimal solutions of DSP_k. In most cases, these modifications break the structure of the dual subproblem exploitable by an efficient combinatorial algorithm. This adds further nuisance to the need of solving an additional linear program. Papadakos (2008) addresses this issue and presents a modified procedure that does not require solving an additional linear program. The modified dual subproblem uses a point that must satisfy characteristics that are more relaxed than Magnanti and Wong's conditions. He also presents a modification to the Benders algorithm to ensure its convergence.

In our algorithm, we use the "tailored" subproblem for MUFND as presented in Magnanti et al. (1986). The authors point out that the additional linear program for each commodity $k \in K$ in the classic Pareto-optimal approach is equivalent to solving the following *parametric minimum cost flow problem*:

$$(\text{MCF}_k) \quad \text{minimize} \sum_{(i,j)\in A} W^k c_{ij}^k x_{ij}^k - DSP_k(\bar{y})x_0 \tag{13}$$

$$\text{s.t.} \quad \sum_{j\in N} x_{ji}^k - \sum_{j\in N} x_{ij}^k = \begin{cases} -(1+x_0) & \text{if } i = o_k \\ 1+x_0 & \text{if } i = d_k \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N \tag{14}$$

$$x_{ij}^k \leq y_{ij}^0 + x_0 \bar{y}_{ij} \qquad \forall (i,j) \in A \tag{15}$$

$$x_{ij}^k \geq 0 \qquad \forall (i,j) \in A$$

$$x_0 \in \mathbb{R}.$$

The problem can be interpreted as that in which a rebate of $\text{DSP}_k(\bar{y})$ is given for each additional unit of the commodity routed on the network with demand and capacities defined by (14) and (15), respectively (Magnanti et al. 1986). The authors show that any fixed value $x_0 \geq \sum_{(i,j)\in A} y_{ij}^0$ is optimal for MCF_k, therefore leaving a minimum cost flow problem to be solved for each commodity $k \in K$.

As a result of fixing x_0, it is no longer necessary to solve $\text{DSP}_k(\bar{y})$ since it is now multiplied by a constant in MCF_k. This observation allows to save computational time and solve MCF_k directly as the separation problem rather than solving it as a complementary problem for Pareto-optimal Benders cuts.

Magnanti and Wong (1981) note that the selection of different core points y^0 leads to varied Pareto-optimal cuts. To the best of our knowledge, the question of selecting an adequate y^0 has not been addressed before in the literature. We provide some guidelines for this in the next section.

4.3. Core Point Selection

To solve MCF_k with a fixed x_0 and without having solved $\text{DSP}_k(\bar{y})$, as in our algorithm, some additional conditions on the core point must be imposed. Among these, is the fact that y^0 must not only lie in the interior of the $\{0,1\}^{|A|}$ hypercube but must also define a network through which one unit of demand can be sent from o_k to d_k, $\forall k \in K$. Failure to do so could lead to MCF_k being infeasible despite $\bar{y}$ being a feasible solution for MUFND. This was observed empirically to have a particularly pernicious effect on the overall computation time.

Preliminary computational tests showed that core point selection significantly influences the solution time leading to savings of up to an order of magnitude for difficult instances as seen in Appendix B of the online supplement. Several strategies for core point selection were tested.

- One is initializing the core point at $\{1\}^{|A|}$ and dynamically updating it as the midpoint between the current solution of MP and the previous iteration's core point as in Papadakos (2008). This strategy has the drawback of sometimes yielding infeasible core points and hence infeasible MCF_k.

- Another is initializing the core point at $\{1\}^{|A|}$ and updating the core point to be the latest incumbent solution of the enumeration tree. This strategy leads to generating more Benders cuts and exploring a larger number of nodes in the enumeration tree.

- The best performing strategy of those tested is to fix the core point throughout the entire process based on the following observed property for our testbed. Approximately 90% of the arcs open in an optimal solution are within the set defined by the union of shortest paths of each commodity $k \in K$. With this in mind, we define a core point that gives higher importance to promising variables, i.e. those that are present in at least one of the commodities' shortest path denoted as $\cup_{k\in K} P_k$. Our core point is initialized as $y^0_{ij} = 0.7$ if $(i,j) \in \cup_{k\in K} P_k$ and $y^0_{ij} = 0.2$ if $(i,j) \notin \cup_{k\in K} P_k$.

These specific values were chosen after running preliminary experiments on our testbed with the values of $\{0.5, 0.6, 0.7, 0.8, 0.9\}$ for arcs in the routing solution and $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ for arcs not in the routing solution. Note that y^0 is in the interior of the $\{0,1\}^{|A|}$ hypercube.

There is no guarantee, however, that y^0 satisfies the feasibility condition previously mentioned. To remedy this we solve a minimum cut problem for each $k \in K$ to check for feasibility. If there exists a minimum cut $\delta(S,\bar{S})_k = \{(i,j) \in A | o_k \in S \text{ and } d_k \in \bar{S}\}$ for a commodity $k \in K$ with $\sum_{(i,j)\in\delta(S,\bar{S})_k} y^0_{ij} < 1$, we then increase the value of each arc in $\delta(S,\bar{S})_k$ by $[1/|\delta(S,\bar{S})_k|]+0.01$ and check again for a violated cutset. This is repeated until no such cutset exists. We place a cap on the value of y^0_{ij} to be at most 0.9999 to ensure y^0 remains an interior point. Given that the core point is fixed throughout the solution process, this verification is only done once at the beginning.

4.4. Benders Lift-and-Project Cuts

With the current trend of implementing Benders decomposition within a branch-and-cut framework, the need for problem formulations to have a strong LP relaxation has become more important. Unfortunately, several problems do not satisfy this property. Recently, Bodur and Luedtke (2017) and Bodur et al. (2017) have proposed the use of general mixed integer cuts such as mixed integer rounding and split cuts, respectively, within a Benders-based branch-and-cut to improve the LP relaxation. Their results show a significant decrease in the LP gap, leading to faster solution times.

Although Hellstrand et al. (1992) show that the polytope defined by P_{UNDP} is quasi-integral, the corresponding Benders reformulation cannot take advantage of this, leaving us to deal with instances that have an LP gap strictly greater than 0. In our method, we adopt the lift-and-project cuts proposed by Balas et al. (1993) to strengthen the master problem LP relaxation.

Lift-and-project cuts, a result of disjunctive programming theory (Balas 1979), were initially proposed as a cutting plane algorithm by Balas et al. (1993) but were later extended to the branch-and-cut framework (Balas et al. 1996) upon proving the ability to make globally valid cuts obtained in the enumeration tree by means of a closed form lifting procedure. The framework is as follows.

Given $P = \{x \in \mathbb{R}^n | \tilde{A}x \geq \tilde{b}\}$ with inequalities of the form $1 \geq x \geq 0$ included in $\tilde{A}x \geq \tilde{b}$ and $P_D = conv\{x \in P | x_j \in \{0,1\}, \forall j = 1...p\}$ where $p < n$, lift-and-project cuts can be obtained by:

1 Selecting an index $\hat{\jmath} \in \{1...p\}$. Multiplying $\tilde{A}x \geq \tilde{b}$ by $(1 - x_{\hat{\jmath}})$ and $x_{\hat{\jmath}}$.

2 Linearizing the obtained system by substituting $y_i = x_{\hat{\jmath}} x_i$ and $x_i = x_i x_i$.

3 Projecting the system back into the original space by means of a cone projection.

Balas (1979) shows that it is possible to obtain the "deepest" lift-and-project cut of the form $\sum_{i \in I} \alpha_i x_i \geq \beta$ that cuts off the LP optimum $\bar{x}$ for a given $\hat{\jmath} \in \{1...p\}$ by solving the following linear program:

$$
\begin{aligned}
(\text{CGLP}_{\hat{\jmath}}) \qquad & \min \sum_{i \in I} \alpha_i \bar{x}_i - \beta \\
\text{s.t.} \quad & \alpha - uA + u^0 e_{\hat{\jmath}} \geq 0 \\
& \alpha - vA - v^0 e_{\hat{\jmath}} \geq 0 \\
& -\beta + ub = 0 \\
& -\beta + vb + v_0 = 0 \\
& u, v \geq 0,
\end{aligned}
$$

where $e_{\hat{\jmath}}$ is the vector of all 0s except for a 1 in the $\hat{\jmath}$-th component.

The feasible space to $\text{CGLP}_{\hat{\jmath}}$ is a convex cone. Therefore, an additional normalization constraint must be added to ensure a finite optimal solution. It has been shown (Balas

and Perregaard 2002) that varied normalizations lead to significantly different cuts. In our implementation, we use the following normalization constraint:

$$\sum_i u_i + u^0 + \sum_i v_i + v^0 = 1.$$

The resulting cut can be strengthened by using a closed formula derived by imposing integrality constraints of other $\{0,1\}$ variables (Balas and Perregaard 2002). We adopt this strengthening procedure in our algorithm. For a given variable x_k, its coefficient α_k in the lift-and-project cut can be replaced by $\alpha'_k = \min\{ua_k + u^0\lceil m_k \rceil, va_k - v^0\lfloor m_k \rfloor\}$, where

$$m_k = \frac{va_k - ua_k}{u^0 + v^0}.$$

In our Benders reformulation we do not have the complete description of the LP relaxation since there would be exponentially many constraints. To circumvent this, we resort to defining $\tilde{A}$ as the feasibility and optimality cuts that are binding at the LP relaxation and the $1 \geq x \geq 0$ constraints. Although this is known to give a weaker lift-and-project cut, we also highlight its role in significantly reducing the size of $CGLP_{\hat{j}}$ leading to times of less than 0.02 seconds to obtain a cut. A similar strategy is used by Balas and Perregaard (2002) outside the context of Benders decomposition. Another important factor in the lift-and-project process is selecting the index $\hat{j}$. In our implementation we choose the fractional variable with highest fixed cost.

During our experiments we noted that these lift-and-project cuts are very dense and often with small, numerically unstable coefficients. This led to numerical issues when too many were added to the master problem at the same time. To circumvent this, we ensure that at most seven lift-and-project cuts are added. We use an additional stopping criterion of increase in the LP optimal value. In other words, if the effect of adding a lift-and-project cut is negligible, we then stop generating them. Note that the effect of the lift-and-project cut is highly dependent on the variable $\hat{j}$ chosen. However, we found this criterion to be an effective rule to prevent numerical instabilities in our algorithm.

4.5. A Slope Scaling Metaheuristic

An important factor in solving difficult optimization problems, in particular when using branch-and-bound methods, is obtaining high quality feasible solutions. Finding these early in the enumeration process often leads to smaller search trees since they provide better bounds for pruning and a guide for selecting variables to branch on. Another use is in a preprocessing stage to eliminate variables that will not be in an optimal solution, as in Contreras et al. (2011). With this in mind, we develop a slope scaling with dual perturbations and long term memory metaheuristic similar to that presented in Crainic et al. (2004) for the capacitated multicommodity fixed-charge network design problem. This metaheuristic is used as a warm start procedure to obtain a high quality solution with which we begin the branch-and-bound process.

Slope scaling was first presented in Yaged (1971) and later in Kim and Pardalos (1999, 2000) as a heuristic to solve network optimization problems. Crainic et al. (2004) improved it by adding Lagrangean perturbation and long term memory to help in diversifying and intensifying the search. The method is based on the idea that there exists a linear program of the form

$$\text{(SS)} \quad \min \sum_{k \in K} \sum_{(i,j) \in A} \hat{c}_{ij}^k x_{ij}^k \tag{16}$$

$$\text{s.t.} \quad \sum_{j \in N} x_{ji}^k - \sum_{j \in N} x_{ij}^k = \begin{cases} -W^k & \text{if } i = o_k \\ W^k & \text{if } i = d_k \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N, \forall k \in K \tag{17}$$

$$0 \leq x_{ij}^k \leq W^k \qquad \forall (i,j) \in A, k \in K, \tag{18}$$

that obtains the same optimal solution as P. The algorithm attempts to estimate the $\hat{c}_{ij}^k$ for which this equivalence holds by defining it as $\hat{c}_{ij}^k = c_{ij}^k + \rho_{ij}^k$, where ρ_{ij}^k is a slope scaling factor that estimates the contribution of the fixed costs. An initial ρ_0 is chosen to begin the algorithm. At each iteration t, SS($\rho(t)$) is solved and its solution is used to obtain ρ_{t+1}.

For our implementation, SS is split into $|K|$ shortest path problems with arc lengths of $\hat{c}_{ij}^k$. We use a multi-start method with different initial values of ρ based on the fixed cost and demand quantity. Let $\bar{W}^k = \sum_{k \in K} W^k / |K|$, i.e. the average demand quantity of the commodities. The initial values used are

- $f_{ij} / \sum_{k \in K} W^k$: the fixed cost divided by the total demand quantity;
- $f_{ij} / \max_{k \in K} W^k$: the fixed cost divided by the largest demand quantity;
- $f_{ij} / \bar{W}^k$: the fixed cost divided by the average demand quantity;
- $f_{ij} / [(\max_{k \in K} W^k - \bar{W}^k)/2]$: the fixed cost divided by the mid-point between the average and maximum demand quantities;
- $f_{ij} / [(\sum_{k \in K} W^k - \bar{W}^k)/2]$: the fixed cost divided by the mid-point between the average and total demand quantities;
- $f_{ij} / [(\sum_{k \in K} W^k - \max_{k \in K} W^k)/2]$: the fixed cost divided by the mid-point between the total and maximum demand quantities.

Upon obtaining the optimal solution $\tilde{x}$ of SS($\rho(i)$), the slope scaling factor is updated as

$$\rho_{ij}^k(i+1) = \begin{cases} \frac{f_{ij}}{\sum_{k \in K} \tilde{x}_{ij}^k} & \text{if } \sum_{k \in K} \tilde{x}_{ij}^k > 0 \\ \rho_{ij}^k(i) & \text{otherwise.} \end{cases} \tag{19}$$

This process is continued until a given number of iterations (T_{SS}) have been performed. Note that upon solving SS, a feasible solution can be constructed for MUFND by fixing to 1 the arcs through which some flow has been sent and solving $|K|$ shortest path problems over this subgraph. To improve the quality of the solution, we then remove any arcs of the subgraph that have not been used in the shortest path of at least one of the commodities.

When two successive iterations obtain the same solution $\tilde{x}$, then the procedure will not produce any new distinct solutions. This may occur before having performed the T_{SS} iterations. To aid in diversifying the search of our metaheuristic, we implement a perturbation tool similar to that of Crainic et al. (2004). When two successive iterations of solving SS obtain the same optimal value or a determined number of iterations T_{pert} without an improved solution have passed, we then proceed to solving the Benders subproblem using the last feasible solution for MUFND. With the dual variables used to define the Benders cuts, we update the slope scaling factor as $\rho_{ij}^k = -\lambda_j^k + \lambda_i^k + \mu_{ij}^k$ and continue iterating until a maximum number T_{SS} of SS models have been solved.

Finally, we implement a long term memory mechanism in which we keep statistics throughout the history of the search. These statistics are used to update ρ and restart the process. Based on whether the current round of the algorithm produced an improved best solution, we choose to update ρ in such a way to intensify or diversify the search.

The statistics kept for each $(i,j) \in A$ and $k \in K$ up to iteration T are the number of iterations for which $\tilde{x}_{ij}^k > 0$ ($n_{ij}^k(T)$), average number of commodities routed through each arc ($\bar{x}_{ij}(T)$) and maximum number of commodities routed through each arc ($\hat{x}_{ij}^k(T)$).

After performing T_{SS} iterations with the corresponding dual perturbations along the way, we calculate for each $(i,j) \in A$, $v_{ij} = \bar{x}_{ij}(T_{SS})/\hat{x}_{ij}(T_{SS})$ or $v_{ij} = 0$ if $\bar{x}_{ij}(T_{SS}) = 0$. Here, v_{ij} measures the variability of the number of commodities sent through arc (i,j) throughout the last T_{SS} iterations. Hence, $v_{ij} \approx 1$ means the number of commodities sent through this arc has been stable throughout the process, while $v_{ij} \approx 0$ shows high variability or no commodities sent at all. During intensification, variables with stable behavior are favoured while the opposite is done when a diversification step is taken.

An intensification update to ρ is done if in the last cycle, an improved best solution was obtained. Otherwise, a diversification step is taken. A limit of div_{max} and int_{max} diversification and intensification updates, respectively, are applied throughout the algorithm. The updates for each scheme are presented below where $\bar{n}$ and S_n are the average and standard deviation of n_{ij}^k, respectively.

- Normalize $\rho_{ij}^k := \rho_{ij}^k - \min_{(i,j)\in A, k\in K} \rho_{ij}^k$ so $\rho_{ij}^k \geq 0 \ \forall (i,j) \in A, k \in K$.
- To apply the intensification scheme, $\forall (i,j) \in A, k \in K$
  - If $n_{ij}^k \geq \bar{n} + S_n$ then $\rho_{ij}^k := \rho_{ij}^k(1 - v_{ij})$
  - If $n_{ij}^k \leq \bar{n}$ then $\rho_{ij}^k := \rho_{ij}^k(2 - v_{ij})$
  - Else $\rho_{ij}^k := \rho_{ij}^k$.
- To apply the diversification scheme, $\forall (i,j) \in A, k \in K$

– If $n_{ij}^k \geq \bar{n} + S_n$ then $\rho_{ij}^k := \rho_{ij}^k(1 + v_{ij}^k)$
– If $n_{ij}^k \leq \bar{n}$ then $\rho_{ij}^k := \rho_{ij}^k(v_{ij}^k)$
– Else $\rho_{ij}^k := \rho_{ij}^k$.

Upon performing some preliminary experiments, our heuristic is run with the parameter values $T_{SS} = 30, T_{DP} = 5, div_{max} = 10, int_{max} = 5$. Below we present a flow chart summarizing the slope scaling with dual perturbation and long term memory sequentially applied to each of the six initial ρ values.

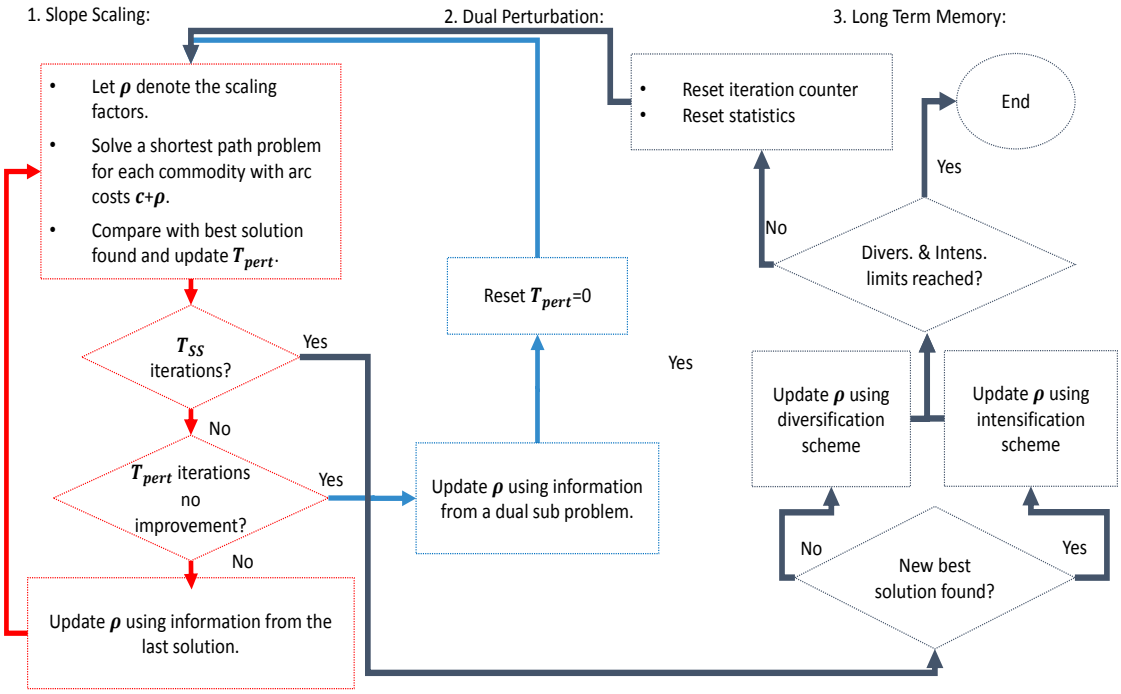

Figure 2: Slope scaling heuristic for MUFND

4.6. Fine-tuning Parameters

We begin our solution process by solving the LP relaxation of MP using our cutset separating routines and MCF_k as our separation oracle to obtain Pareto-optimal Benders cuts with core points defined as in Section 4.3. Upon confirming that no more violated Benders cuts exist, we use those that are binding at the optimal solution to obtain a violated lift-and-project cut. This cut is added to the MP relaxation and we resume separating violated Benders cuts. We repeat this process until our stopping criteria are satisfied.

We then define the MIP problem in CPLEX with the active Benders and lift-and-project cuts where the active Benders constraints are added as lazy and user constraints. This prevents defining an excessively large initial problem. Another of the important aspects to consider when implementing this method is the separation and cut adding

frequency. Adding too few cuts leads to an underestimation of the lower bounds of nodes in the enumeration tree, while adding too many cuts leads to large LPs that require a longer computation time to solve.

Several cutting frequencies were tested in preliminary experiments. The best of the tested strategies was separating Benders cuts at all nodes in the first five levels of the enumeration tree and then separating at every 100th node. For all these, only one round of violated Benders cuts are added. For fractional solutions, a minimum violation of 0.01 was required to add the cut to the constraint pool of the node's linear problem. Our last fine-tuning detail is setting CPLEX's MIP Emphasis parameter to optimality given that we provide CPLEX with an initial feasible solution.

5. A Cut-and-Solve Algorithm for MUFND

Introduced by Climer and Zhang (2006) in the artificial intelligence community, cut-and-solve has been used in solving well-known combinatorial optimization problems such as the *travelling salesman problem* and the *single-source capacitated facility location problem* (Yang et al. 2012, Gadegaard et al. 2017). The cut-and-solve framework can be thought of as a generalized local branching (Fischetti and Lodi 2003) in which at each level of the enumeration tree only two child nodes exist, one corresponding to a smaller "sparse" problem and the other as its complement known as the "dense" problem. Unlike local branching where the definition of each are explicitly given by an initial solution and a Hamming distance restriction, cut-and-solve allows for more generic problem definitions. The "sparse" and "dense" problems are defined by constraints over a set of variables. These constraints, known as "piercing" cuts, are of the form $\sum_{i \in I} x_i \leq \sigma$ and $\sum_{i \in I} x_i \geq \sigma + 1$ where $I \subset N$ is a subset of the problem's binary variables and $\sigma \in \mathbb{Z}$.

Upon branching, the "sparse" problem is solved to optimality by means of branch-and-bound or any exact method to obtain a primal bound (UB_{sparse}) on the original problem. This highlights the aim to define sparse problems that are easily solved. Next, the linear relaxation of the dense problem is solved to obtain a lower bound (LB_{dense}) on the remaining solution space of the original problem. If $LB_{dense} \geq UB_{sparse}$ then UB_{sparse} is optimal for the complete problem. Otherwise another piercing cut is defined over the dense problem and the procedure is repeated.

We propose the use of our Benders-based branch-and-cut as the black box MIP solver within the cut-and-solve algorithm. Two important advantages of using our Benders algorithm within the cut-and-solve framework are the reduced problem size and the reusability of the Benders cuts generated in previous sparse problems. On the other hand, some advantages to using cut-and-solve over Benders is that piercing cuts significantly reduce the solution space and the optimal values of previous sparse problems are useful for pruning branches in the enumeration tree. To the best of our knowledge, the idea of combining these methods is new and provides an interesting research direction for accelerating these solution techniques.

Our cut-and-solve algorithm begins by constructing a feasible solution to MUFND. We use the union of shortest paths of each commodity considering only transportation

costs, as in our core point definition. Although this solution does not provide a good upper bound, it does however contain a large percentage of arcs that are in the optimal solution.

For ease of exposition we introduce the following notation. Let $(\bar{y}, \bar{z})(t)$ represent the solution of the t-th sparse problem, $I(t)$ denote the set of indices of arc variables whose value is 1 in $(\bar{y}, \bar{z})(t)$ and $\chi(\bar{y}, \bar{z})(t))$ be its objective function value. In particular, $(\bar{y}, \bar{z})(0)$ refers to the initial feasible solution from which we begin the cut-and-solve process.

At a given $t \geq 1$, we define the following sparse problem:

$$(\mathrm{MP}_{sparse}(t)) \quad \min \sum_{(i,j)\in A} f_{ij}y_{ij} + \sum_{k\in K} z_k \tag{20}$$

$$\text{s.t.} \qquad z_k \geq \lambda^k_{d_k} - \lambda^k_{o_k} - \sum_{(i,j)\in A} \mu^k_{ij} y_{ij} \qquad \forall (\lambda, \mu)_k \in Opt(DSP_k), \forall k \in K \tag{21}$$

$$\sum_{(i,j)\in\delta(S,N\setminus S)} y_{ij} \geq 1 \qquad \forall S \in \Omega \tag{22}$$

$$z_k \geq 0 \qquad \forall k \in K \tag{23}$$

$$y \in \{0,1\}^{|A|} \tag{24}$$

$$\sum_{(i,j)\notin I(t-1)} y_{ij} \leq t \tag{25}$$

$$\sum_{(i,j)\notin I(s)} y_{ij} \geq s+2 \qquad \forall s = 0, ..., t-1 \tag{26}$$

$$\sum_{(i,j)\in A} f_{ij}y_{ij} + \sum_{k\in K} z_k \leq \chi((\bar{y}, \bar{z})(t-1)), \tag{27}$$

where $\Omega = \{S \subset N | \exists k \in K \text{ where } o_k \in S, d_k \notin S\}$. The constraints (20)-(24) are the Benders master problem reformulation of MUFND. Constraint (25) is the piercing cut that allows at most t variables not in the previous solution to take the value of 1 while constraints (26) are the negations of (25) from previous iterations. The latter ensure that previously searched areas of the feasible space are not considered in the new sparse problem. Finally, constraint (27) imposes that the optimal solution of the current sparse problem has objective value of at most the optimal value of the previous one. This constraint ensures that the obtained solutions do not worsen after each iteration and saves computation time since its value is used as a pruning criterion for the enumeration tree.

If $\mathrm{MP}_{sparse}(t)$ is feasible, we define $\mathrm{MP}_{sparse}(t+1)$ without solving the LP relaxation of a corresponding dense problem. In this respect, our algorithm bears resemblance to local branching as proposed in Fischetti and Lodi (2003). This is done until two successive optimal solutions to $\mathrm{MP}_{sparse}(t)$ are the same or until $\mathrm{MP}_{sparse}(t)$ is infeasible, due to (27). If either occurs, the dense problem, MP_{dense}, is defined. The dense problem is similar to $\mathrm{MP}_{sparse}(t)$ with the exception that (25) is replaced by $\sum_{(i,j)\notin I(t-1)} y_{ij} \geq t+1$ and $(\bar{y}, \bar{z})(t-1)$ in (27) is replaced by the best solution found so far. We then solve

$\text{MP}_{dense}(t)$ which will either be feasible and hence provide the true optimal solution or will be infeasible meaning that the best solution found so far is indeed optimal. Below we present the pseudocode of our cut-and-solve Benders algorithm.

Algorithm 1 Cut-and-solve Benders algorithm for MUFND

Require: 0: Initialization
  $(\bar{y}, \bar{z})(0) = \cup_{k \in K} P_k$, $t = 1$, $best = (\bar{y}, \bar{z})(0)$
  Step 1: Define and solve $\text{MP}_{sparse}(t)$
  if ($\text{MP}_{sparse}(t)$ is feasible $\wedge$ $(\bar{y}, \bar{z})(t-1) \neq (\bar{y}, \bar{z})(t)$) **then**
    $best = (\bar{y}, \bar{z})(t)$
    $t = t + 1$;
    `Goto` **Step 1**
  else
    `Goto` **Step 2**
  end if
  Step 2: Define and solve MP_{dense}
  if MP_{dense} is feasible **then**
    Update *best*
  end if
  Return *best*.

6. Computational Experiments

We perform extensive computational experiments to evaluate the efficiency of our proposed methods and the effect of the enhancements implemented. Our analyses focus on: the LP gap closed by adding lift-and-project cuts to our master problem linear relaxation, the quality of solutions obtained from our metaheuristic, and the efficiency of our proposed solution methods versus the state-of-the-art general-purpose MIP solver CPLEX 12.7.1 using both its default branch-and-cut and its new Benders decomposition feature.

We use the well-known "Canad" multicommodity capacitated network design instances (Crainic et al. 2001) as our testbed. This dataset consists of 205 instances with arc capacities. Ignoring the capacity constraints leaves a total of 93 distinct instances for our experiments. The testbed can be divided into three classes. The first are the 31 "C" instances with many commodities compared to nodes while the second are eight "C+" instances with few commodities compared to nodes. Finally, Class III is divided into two subgroups. Class III-A and III-B are each comprised of 27 "R" instances on small and medium sized graphs, respectively.

We generate eight large-scale instances, denoted as Class IV, on which we test our algorithms with a 24-hour time limit. These were generated using the Mulgen generator (Crainic et al. 2001) available at *http://pages.di.unipi.it/frangio/* with sizes of up to 1,500 arcs and 1,500 commodities. To the best of our knowledge these are the largest instances of the MUNDP to be solved by an exact algorithm.

Another characteristic of our testbed is the existence of instances that have an LP gap strictly greater than 0. This is important in our analysis as we need to test our algorithm's ability to quickly explore the enumeration tree and the efficiency of our proposed lift-and-project Benders cuts. The "Canad" testbed contains several instances with this property. Table 2 details the number of instances for each LP gap range (%) in our testbed.

Table 2: Distribution of "Canad" instances' LP gaps (%)

Class	**0**	$(0,1]$	$(1,2]$	$(2,3]$	$(3,4]$	$[4,7.2]$	**Total**
Class I	10	7	2	4	2	6	31
Class II	7	1					8
Class III-A	26		1				27
Class III-B	9	2		2	5	9	27
Class IV	0	3	3	2			8
Total	**52**	**13**	**6**	**8**	**7**	**15**	**101**

All algorithms were coded in C using the callable library for CPLEX 12.7.1. The separation and adding of cutset inequalities and Benders optimality cuts is implemented via lazycallbacks and usercutcallbacks. For a fair comparison, all use of CPLEX was limited to one thread and the traditional MIP search strategy. Experiments were executed on an Intel Xeon E5 2687W V3 processor at 3.10 GHz under Linux environment.

6.1. Impact of Lift-and-Project Cuts on LP Gap

As shown in Table 2, Class I, III-B, and IV contain most of the instances with higher LP gap. Preliminary tests showed these to be the most difficult to solve, in particular when it came to proving optimality. It is in this spirit that we proposed using lift-and-project cuts to improve the LP bound at the root node.

The following table shows the percentage of the LP gap ($LP_{imp}\%$) closed by applying at most seven lift-and-project cuts to the linear relaxation of our Benders master problem. This percentage is calculated as $LP_{imp} = 100 \times \frac{(LP_{MPLP} - LP_{MP})}{Opt - LP_{MP}}$ where LP_{MPLP} is the optimal value of the linear relaxation of the master problem with the additional lift-and-project cuts, LP_{MP} is the optimal value of the linear relaxation of the master problem, and Opt is the optimal value of the problem.

Table 3: LP gap (%) closed

Class	**No. of instances**	LP_{imp}
Class I	21	6.61
Class II	1	9.07
Class III-A	1	20.45
Class III-B	18	3.65
Class IV	8	2.99
Grand Total	**49**	**5.26**

The average improvement of the LP gap is of 5.26% over the 50 solved instances with an LP gap. There are many factors that contribute to this behavior. The first is that lift-and-project cuts as proposed by Balas et al. (1993) require the complete formulation of the problem. In our implementation, we use a relaxation comprised of only the Benders cuts that are binding at the LP solution. In Balas et al. (1996), this relaxation is shown to generate weaker cuts. Second, our stopping criterion for lift-and-project cuts is conservative, avoiding generating too many of them at the beginning due to their numerical instability. Finally, our simplified variable selection rule also contributes to this performance.

6.2. Solution Quality of Primal Heuristic

We show the primal heuristic's solution quality and computation time. Table 4 presents for each class and size of our testbed, the number of instances in each description (Nm), the average relative gap ($\bar{rel}_{gap}$) of the metaheuristic's solution compared to the optimal value calculated and the average CPU time in seconds (s) where the relative gap is calculated as $rel_{gap} = 100 \times (UB_{heur} - Opt)/Opt$.

Table 4: Solution quality of our metaheuristic

Class I				Class II			
Description	Nm	$\overline{rel}_{gap}$	s	Description	Nm	$\overline{rel}_{gap}$	s
20,230,40	3	0.08	2.94	25,100,10	2	1.01	0.59
20,300,40	4	0.56	3.61	25,100,30	2	0.48	1.56
20,230,200	4	2.98	7.56	100,400,10	2	2.77	3.83
20,300,200	4	2.81	8.81	100,400,30	2	0.82	12.16
30,520,100	4	4.16	11.42				
30,520,400	4	1.59	42.34				
30,700,100	4	2.83	14.11				
30,700,400	4	1.39	51.7				
Sub-Total	**31**	**2.11**	**18.29**	**Sub-Total**	**8**	**1.27**	**4.53**
Class III-A				Class III-B			
Description	Nm	$\overline{rel}_{gap}$	s	Description	Nm	$\overline{rel}_{gap}$	s
10,35,10	3	0	0.32	20,120,40	3	0.46	1.44
10,35,25	3	0.06	0.61	20,120,100	3	1.55	3.51
10,35,50	3	0	1.11	20,120,200	3	1.46	5.85
10,60,10	3	0	0.36	20,220,40	3	2.04	1.94
10,60,25	3	0.83	0.67	20,220,100	3	2.82	3.64
10,60,50	3	0.49	1.2	20,220,200	3	2.64	7.04
10,85,10	3	0.08	0.37	20,320,40	3	4.58	2.21
10,85,25	3	0.51	0.73	20,320,100	3	2.72	4.87
10,85,50	3	0.47	1.13	20,320,200	3	4.55	8.7
Sub-Total	**27**	**0.27**	**0.72**	**Sub-Total**	**27**	**2.54**	**4.35**
Class IV							
Description	Nm	$\overline{rel}_{gap}$	s				
40,1200,400	1	1.93	102.5				
40,1200,800	1	1.02	191.25				
40,1200,1200	1	1.34	319.77				
50,1500,400	1	2.09	110.63				
50,1500,800	1	2.01	222.78				
50,1500,1200	1	1.97	363.15				
50,1500,1000	1	1.85	379.09				
50,1500,1500	1	2.94	625.36				
Sub-Total	**8**	**1.90**	**289.32**				
Total	**101**	**1.65**	**30.25**				

The proposed slope scaling/dual perturbation/long term memory algorithm performs reasonably well over our 101 instances, many of which have an LP gap strictly greater than 0. The solutions obtained are on average 1.65% away from the optimal solution in an average of 7.6 seconds of CPU time for Classes I, II and III and 290 seconds for the large-scale instances. Table 5 details the number of instances in each optimality gap (%)

range when applying our primal heuristic.

Table 5: Distribution of optimality gap (%)

Class	**0**	**(0,2]**	**(2,4]**	**(4,6]**	**(6,7]**	**Total**
I	1	18	7	4	1	31
II	4	2	1	1		8
III-A	17	10				27
III-B	1	11	9	6		27
IV	0	5	3	0		8
Total	**23**	**46**	**20**	**11**	**1**	**101**

As seen, our heuristic obtains the optimal solution for 23 instances, mostly from Class III-A. This is expected in particular since all but one instance in the class have no LP gap, thereby favoring the use of the slope scaling heuristic. On the other hand, for 88% of the instances we obtain solutions that are between 0 and 4% away from the optimal solution, including most of those that have an LP gap strictly greater than 0. The instance for which our metaheuristic has the worst performance finds a solution 6.35% away from the optimal value. Overall, our primal heuristic gives promising results and seems to provide a reasonable starting point for our Benders-based branch-and-cut.

6.3. Computation Time

We now compare the computation time of each of our proposed algorithms. We begin by focusing on our Benders-based branch-and-cut since we use the best performing as the black box solver in our cut-and-solve/local branching algorithm. To show the impact of each enhancement, we present four versions of our Benders-based branch-and-cut. The first is without using our primal heuristic nor our lift-and-project cuts (B_0). The second is the same, with an initial solution provided by our primal heuristic (B_1). B_2 is the Benders-based branch-and-cut with lift-and-project cuts added at the root node and the final version (B_3) combines them all.

Table 6: Computational performance of Benders branch-and-cut

		B_0		B_1		B_2		B_3	
Class	**Description**	**(s)**	**Nodes**	**(s)**	**Nodes**	**(s)**	**Nodes**	**(s)**	**Nodes**
I	20,230,40	0.15	0	4.55	0	0.15	0	4.56	0
	20,230,200	13.46	274.5	23.97	333	19.16	331.75	23.56	281
	20,300,40	0.17	0.25	4.58	0.25	0.2	0.75	4.66	0.5
	20,300,200	10.75	200.5	16.4	161.25	12.67	220	19.77	235.75
	30,520,100	300.61	3335.25	189.6	2094.25	168.1	2390.75	631.19	5491.75
	30,520,400	6.78	35.25	43.52	34.25	7.26	45.5	44.46	34.5
	30,700,100	7.99	188.25	19.9	193.25	9.97	346.25	25.27	334.25
	30,700,400	321.09	4613.25	396.29	4619.75	389.88	7047.25	792.12	9970.25
	Sub-Total	**85.29**	**1115.77**	**90.02**	**959.48**	**78.37**	**1339.65**	**199.28**	**2109.42**
II	25,100,10	0.02	0	0.97	0	0.02	0	0.97	0
	25,100,30	0.05	0	2.65	0	0.05	0	2.66	0
	100,400,10	9.12	0	12.58	0	8.85	0	12.51	0
	100,400,30	17.05	4.5	31.68	8.5	17.39	7.5	31.54	5.5
	Sub-Total	**6.56**	**1.13**	**11.97**	**2.13**	**6.57**	**1.88**	**11.92**	**1.38**
III-A	10,35,10	0.01	0	0.76	0	0.01	0	0.76	0
	10,35,25	0.02	0	1.55	0	0.01	0	1.56	0
	10,35,50	0.03	0	2.92	0	0.03	0	2.92	0
	10,60,10	0.02	0	0.84	0	0.01	0	0.86	0
	10,60,25	0.02	0	1.52	0	0.02	0	1.53	0
	10,60,50	0.03	0	2.77	0	0.04	0	2.8	0
	10,85,10	0.01	0	0.8	0	0.01	0	0.81	0
	10,85,25	0.02	0	1.54	0	0.02	0	1.56	0
	10,85,50	0.08	3.33	2.33	4.67	0.12	4.67	2.39	4.33
	Sub-Total	**0.03**	**0.37**	**1.67**	**0.52**	**0.03**	**0.52**	**1.69**	**0.48**
III-B	20,120,40	0.07	0	2.29	0	0.08	0	2.26	0
	20,120,100	1.64	70.67	7.75	98	3.98	225.33	9	232
	20,120,200	208.51	1808	252.19	1808.67	224.44	3254.33	209.42	2055.67
	20,220,40	1.73	100.67	3.97	96.33	1.73	112.33	4.91	142.67
	20,220,100	37.93	688.67	39.32	623	173.4	2555.67	94.5	1510
	20,220,200	1019.59	7831	1271.31	6495.33	1616.01	10740.33	1503.22	9482.33
	20,320,40	10.88	876.67	16.99	824.67	12.14	1116.33	19.21	1224
	20,320,100	6.17	104.33	15.43	140	9.54	226.33	17.91	250.67
	20,320,200	810.14	4192.33	1166.33	6812.33	1548.18	7692	2032.04	9906.67
	Sub-Total	**232.96**	**1741.37**	**308.4**	**1877.59**	**398.83**	**2880.3**	**432.5**	**2756**
IV	40,1200,400	6.75	6	103.48	5	9.31	12	105.77	5
	40,1200,800	38.28	537	227.45	712	53.65	902	237.28	668
	40,1200,1200	42.83	61	348.82	54	53.2	36	357.98	54
	50,1500,400	21.43	649	140.38	790	22.96	556	139.63	563
	50,1500,800	95.45	1938	342.28	4285	108.36	2799	307.43	1964
	50,1500,1200	163.14	2162	506.31	1619	200.54	2375	546.6	2488
	Sub-Total	**61.31**	**892.17**	**278.12**	**1244.17**	**74.67**	**1113.33**	**282.45**	**957**
	Total	**94.49**	**878.57**	**130.57**	**888.23**	**138.38**	**1272.79**	**198.9**	**1470.4**

We note that with respect to computation time, implementing Benders without an initial solution nor lift-and-project cuts performs on average the fastest, followed by warm starting with a high quality initial solution and finally enhancing it with lift-and-project cuts. The difference between using or not an initial solution to warm start the branch-and-cut process is not negligible. In fact, the results show it is preferable to not provide an initial solution, in particular for the large-scale instances of Class IV. On the other hand, the computation time is almost doubled when including lift-and-project cuts at the

root node both when applied alone or in combination with an initial solution.

One explanation for both observations is the role that branching decisions play in the solution process. While good initial solutions provide useful bounds for pruning and a guide for branching, ultimately there is no guarantee that these will indeed lead to a smaller search tree and thus faster computation time. Even in the event that the optimal solution is given to the branch-and-cut algorithm, there exists an exponential number of ways from which the LP solution can be taken to the optimum provided. These play an important role in determining the size of the search tree and computation time. Our experiments show that the branching decisions taken in our lift-and-project enhanced Benders lead to a significantly larger enumeration tree, overshadowing the benefit obtained from having a smaller LP gap at the root node. Similar behavior is seen for B_1 leading to increased solution times in particular for the larger instances. For these, an important contributing factor is the time needed, on average 30 seconds, to obtain these high quality solutions via our metaheuristic.

The use of an initial solution has the same effect for our lift-and-project enhanced Benders, increasing the solution time needed. As seen in Table 6, the computation time for these versions is almost double that of B_0. An explanation for this is the density and numerical instability of the lift-and-project cuts. As mentioned before, these cuts have several non-zero coefficients close to 0. This leads to more time required to solve the underlying linear programs and in some instances numerical instability that prevents CPLEX from constructing an advanced basis for nodes in the tree.

Given these results, we choose B_0 as the black box solver for our cut-and-solve algorithm. In addition, we focus on the 66 instances comprised by Classes I, III-B, IV. As seen in Table 6, Classes II and III-A are small scale instances that are easily solved due to their size and lack of an LP gap. Figure 3 is the performance profile of our Benders-based branch-and-cut ($BB\&C$), our cut-and-solve (CS/LB) and CPLEX 12.7.1's branch-and-cut algorithm CPX on P. We do not compare with CPLEX's black box Benders implementation since preliminary results showed it to perform significantly worse than CPLEX's branch-and-cut. Figure 3 plots the number of instances solved by each algorithm within a given number of seconds.

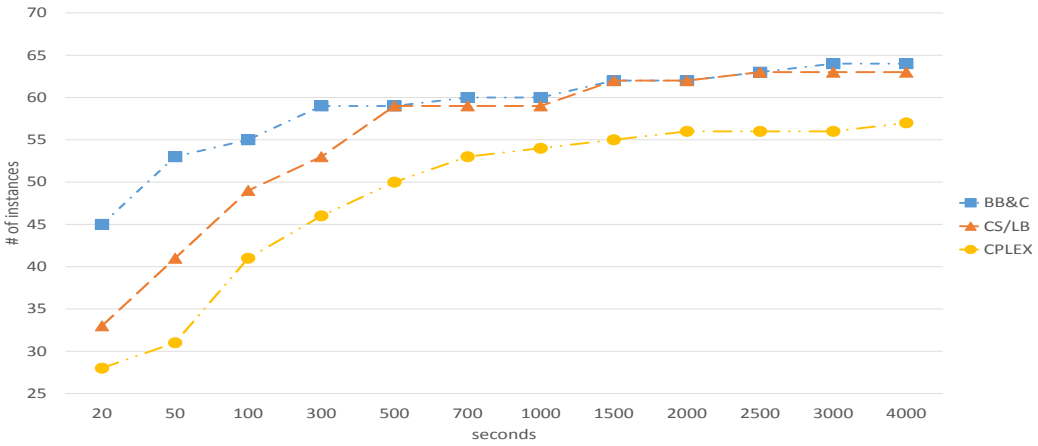

Figure 3: Number of instances solved in a given time limit

We note that both our proposed methodologies are able to solve more instances in less time than CPLEX, having solved close to 83% of the instances in less than 100 seconds and solving all medium sized instances within 45 minutes. CPLEX on the other hand manages to solve only 41 instances within 100 seconds; the same number solved by our Benders-based branch-and-cut method in 20 seconds. CPLEX runs into trouble proving optimality for 10 instances requiring over an hour for the least troublesome and over half a day for the most burdensome. In addition, it times out for two instances. Our Benders-based branch-and-cut solves all instances within 12.2 hours of CPU time.

When comparing the performance between (CS/LB) and $(BB\&C)$, we see from Figure 3 that the latter is faster at solving instances. However, when given additional time, both algorithms have similar behavior. To make a more precise comparison, Table 7 contains the average times required for each class and description of our testbed.

Table 7: Comparison of computational performance

Class	Description	Nb	*BB&C*	*CS/LB*	*CPLEX*
I	20,230,40	3	0.15	0.55	0.07
	20,230,200	4	13.46	43.59	252.95
	20,300,40	4	0.17	0.74	0.17
	20,300,200	4	10.75	39.54	303.24
	30,520,100	4	300.61	331.24	3181.33
	30,520,400	4	6.78	27.29	95.46
	30,700,100	4	7.99	30.87	71.61
	30,700,400	4	321.09	589.39	10479.58
	Sub-Total	**31**	**85.29**	**137.17**	**1856.05**
III-B	20,120,40	3	0.07	0.33	0.05
	20,120,100	3	1.64	5.53	13.42
	20,120,200	3	208.51	362.36	361.23
	20,220,40	3	1.73	7.78	6.91
	20,220,100	3	37.93	58.78	153.86
	20,220,200	3	1019.59	1583.34	1615.31
	20,320,40	3	10.88	25.94	27.79
	20,320,100	3	6.17	185.56	69.25
	20,320,200	3	810.14	561.13	2592.58
	Sub-Total	**27**	**232.96**	**310.09**	**537.82**
IV	40,1200,400	1	7.95	33.43	59.82
	40,1200,800	1	41.4	405.23	4483.75
	40,1200,1200	1	48.87	118.02	1664.1
	50,1500,400	1	23.26	86.04	575.91
	50,1500,800	1	151.85	345.59	39051.12
	50,1500,1200	1	226.64	484.83	57071.73
	50,1500,1000	1	32,402.16	time	time
	50,1500,1500	1	63,171.93	time	time
	Sub-Total	**6**	**83.33**	**245.52**	**17151.07**
	Total	**64**	**147.4**	**220.28**	**2733.83**

Our results show that on average both $BB\&C$ and CS/LB are an order of magnitude faster than CPLEX. This speedup is even more significant when limiting our analysis to the large-scale instances. For these, our Benders-based branch-and-cut is up to two orders of magnitude faster than CPLEX. The instances of Class III-B also show a significant saving in computation time in favor of our Benders-based algorithms. The savings obtained with $BB\&C$ can be largely attributed to solving smaller underlying linear programs in the enumeration tree and exploring the nodes in significantly less time.

This is surprising because unlike CPLEX, our Benders-based branch-and-cut does not explicitly have a complete description of the problem's polytope but must instead estimate it on the fly by generating Benders cuts. This leads to the possibility of underestimating

the solution of the underlying linear programs at each node of the enumeration tree, leading to weak dual bounds. This is more likely to occur in $BB\&C$ since we only allow for one round of Benders cuts to be added at non-root nodes of our enumeration tree. However, due to the enhancements proposed, we leave the root node with a linear program that captures most of the complete formulation's characteristics in a smaller problem.

On the other hand, our cut-and-solve algorithm's performance is also two orders of magnitude faster than CPLEX for large-scale instances while for Class III-B, it saves 40% of the solution time. We also point out that for the Class III-B instances the performance of CS/LB is more similar to $BB\&C$ than in Class I. This shows the robustness of this method over different types of instances; requiring less fine-tuning than its branch-and-cut counterpart. On average, CS/LB solves four sparse problems before proving optimality of its obtained solution. Each of these sparse problems are solved up to three orders of magnitude faster than solving the complete problem with CPLEX and sometimes in half the time than if solved with Benders-based branch-and-cut. It is because of these time savings that it outperforms CPLEX in all instances and our Benders-based branch-and-cut in one of them.

7. Conclusion

We have presented two exact solution algorithms for the multicommodity uncapacitated fixed-charge network design problem that significantly outperform the state-of-the-art general-purpose MIP solver CPLEX. The first exact algorithm is based on implementing Benders decomposition within a branch-and-cut framework using Pareto-optimal cuts, appropriate core point selection and fine-tuned cutting frequency. These additional refinements also serve as general guidelines for implementing this algorithm for other mixed integer problems. We also proposed a slope scaling metaheuristic that obtains solutions that are on average only 1.65% away from the optimal solution.

We present a novel strategy for improving the LP bound of our Benders reformulation by means of lift-and-project cuts applied to the master problem's feasibility and optimality cuts. These are obtained using a modified cut generating linear program that takes less than 0.02 seconds to solve. This procedure extends beyond MUFND to any problem on which Benders decomposition is applied, thereby providing a novel family of valid inequalities for mixed integer programs.

Finally, we present a strategy that combines ideas from cut-and-solve, local branching and our proposed Benders-based branch-and-cut. The advantages of this method are: breaking down the problem into a few sparse MIPs which make it easier to obtain high quality feasible solutions, the non-increasing optimal values obtained from the sparse problems, the reduced size of the sparse problem solution space, and the re-usability of Benders cuts generated in previous iterations. Further research can be done on defining initial solutions and having an improved framework to take advantage of Benders cuts generated in previous sparse problems. Nevertheless, the results of our implementation show this fusion to be a promising method for solving large-scale MIPs.

References

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

Ahuja RK, Magnanti TL, Orlin JB (1991) Some recent advances in network flows. *SIAM Review* 33(2):175–219.

Ahuja RK, Magnanti TL, Orlin JB (1993) *Network Flows: Theory, Algorithms, and Applications* (Prentice-Hall, New Jersey, U.S.A.).

Ahuja RK, Magnanti TL, Orlin JB, Reddy M (1995) Applications of network optimization. Nemhauser GL, Rinnooy Kan AHG, Todd MJ, eds., *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, 1–83.

Andersen J, Crainic TG, Christiansen M (2009) Service network design with management and coordination of multiple fleets. *European Journal of Operational Research* 193(2):377–389.

Assarf B, Gawrilow E, Herr K, Joswig M, Lorenz B, Paffenholz A, Rehn T (2017) Computing convex hulls and counting integer points with polymake. *Mathematical Programming Computation* 9(1):1–38.

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.

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

Balakrishnan N, Wong RT (1990) A network model for the rotating workforce scheduling problem. *Networks* 20(1):25–42.

Balas E (1979) Disjunctive programming. *Annals of Discrete Mathematics* 5:3–51.

Balas E, Ceria S, Cornuéjols G (1993) A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming* 58(1):295–324.

Balas E, Ceria S, Cornuéjols G (1996) Mixed 0–1 programming by lift-and-project in a branch-and-cut framework. *Management Science* 42(9):1229–1246.

Balas E, Perregaard M (2002) Lift-and-project for mixed 0–1 programming: recent progress. *Discrete Applied Mathematics* 123(1):129–154.

Bartholdi JJ, Orlin JB, Ratliff HD (1980) Cyclic scheduling via integer programs with circular ones. *Operations Research* 28(5):1074–1085.

Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4:238–252.

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

Billheimer J, Gray P (1973) Network design with fixed and variable cost elements. *Transportation Science* 7(1):49–74.

Bodur M, Dash S, Günlük O, Luedtke J (2017) Strengthened Benders cuts for stochastic integer programs with continuous recourse. *INFORMS Journal on Computing* 29(1):77–91.

Bodur M, Luedtke JR (2017) Mixed-integer rounding enhanced Benders decomposition for multiclass service-system staffing and scheduling with arrival rate uncertainty. *Management Science* 63(7):2073–2091.

Boffey T, Hinxman A (1979) Solving the optimal network problem. *European Journal of Operational Research* 3(5):386–393.

Botton Q, Fortz B, Gouveia L, Poss M (2013) Benders decomposition for the hop-constrained survivable network design problem. *INFORMS Journal on Computing* 25(1):13–26.

Climer S, Zhang W (2006) Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence* 170(8):714–738.

Contreras I, Cordeau JF, Laporte G (2011) Benders decomposition for large-scale uncapacitated hub location. *Operations research* 59(6):1477–1490.

Cordeau JF, Pasin F, Solomon MM (2006) An integrated model for logistics network design. *Annals of Operations Research* 144(1):59–82.

Cordeau JF, Stojković G, Soumis F, Desrosiers J (2001) Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation Science* 35(4):375–388.

Costa AM (2005) A survey on Benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research* 32(6):1429–1450.

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

Crainic TG (2000) Service network design in freight transportation. *European Journal of Operational Research* 122(2):272–288.

Crainic TG, Frangioni A, Gendron B (2001) Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics* 112(13):73–99.

Crainic TG, 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.

Crainic TG, Rousseau JM (1986) Multicommodity, multimode freight transportation: A general modeling and algorithmic framework for the service network design problem. *Transportation Research Part B: Methodological* 20(3):225–242.

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

Dionne R, Florian M (1979) Exact and approximate algorithms for optimal network design. *Networks* 9(1):37–59.

Edmonds J, Karp RM (1972) Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computing Machinery* 19(2):248–264.

Fischetti M, Ljubić I, Sinnl M (2017) Redesigning Benders decomposition for large-scale facility location. *Management Science* 63(7):2146–2162.

Fischetti M, Lodi A (2003) Local branching. *Mathematical Programming* 98(1-3):23–47.

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

Fragkos I, Cordeau JF, Jans R (2017) The multi-period multi-commodity network design problem. Technical Report CIRRELT 2017-63, Université de Montréal.

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):133–161.

Gadegaard SL, Klose A, Nielsen LR (2017) An improved cut-and-solve algorithm for the single-source capacitated facility location problem. *EURO Journal on Computational Optimization.* In press.

Geoffrion AM, Graves GW (1974) Multicommodity distribution system design by Benders decomposition. *Management Science* 26(8):855–856.

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

Ghamlouche I, Crainic TG, Gendreau M (2004) Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research* 131(1):109–133.

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

Hellstrand J, Larsson T, Migdalas A (1992) A characterization of the uncapacitated network design polytope. *Operations Research Letters* 12(3):159–163.

Holmberg K, Hellstrand J (1998) Solving the uncapacitated network design problem by a Lagrangean heuristic and branch-and-bound. *Operations Research* 46(2):247–259.

Johnson DS, Lenstra JK, Kan AHGR (1978) The complexity of the network design problem. *Networks* 8(4):279–285.

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

Kim D, Pardalos PM (2000) Dynamic slope scaling and trust interval techniques for solving concave piecewise linear network flow problems. *Networks* 35(3):216–222.

Kratica J, Tošić D, Filipović V, Ljubić I (2002) A genetic algorithm for the uncapacitated network design problem. Roy R, Köppen M, Ovaska S, Furuhashi T, Hoffmann F, eds., *Soft Computing and Industry: Recent Applications*, 329–336.

Lamar BW, Sheffi Y, Powell WB (1990) A capacity improvement lower bound for fixed charge network design problems. *Operations Research* 38(4):704–710.

Los M, Lardinois C (1982) Combinatorial programming, statistical optimization and the optimal transportation network problem. *Transportation Research Part B: Methodological* 16(2):89–124.

Magnanti T, Mireault P, Wong R (1986) Tailoring Benders decomposition for uncapacitated network design. Gallo G, Sandi C, eds., *Network Flow at Pisa*, 112–154, Mathematical Programming Studies, volume 26.

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

Naoum-Sawaya J, Elhedhli S (2013) An interior-point Benders based branch-and-cut algorithm for mixed integer programs. *Annals of Operations Research* 210(1):33–55.

Ortega F, Wolsey L (2003) A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks* 41(3):143–158.

Ortiz-Astorquiza C, Contreras I, Laporte G (2017) An exact algorithm for multi-level uncapacitated facility location. S*ubmitted to* T*ransportation* S*cience*.

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

Papadakos N (2009) Integrated airline scheduling. *Computers and Operations Research* 36(1):176–195.

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

Rahmaniani R, Crainic TG, Gendreau M, Rei W (2017) 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.

Santoso T, Ahmed S, Goetschalckx M, Shapiro A (2005) A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research* 167(1):96–115.

Yaged B (1971) Minimum cost routing for static network models. *Networks* 1(2):139–172.

Yaghini M, Karimi M, Rahbar M, Sharifitabar MH (2015) A cutting-plane neighborhood structure for fixed-charge capacitated multicommodity network design problem. *INFORMS Journal on Computing* 27(1):48–58.

Yang Z, Chu F, Chen H (2012) A cut-and-solve based algorithm for the single-source capacitated facility location problem. *European Journal of Operational Research* 221(3):521–532.

Appendix A. Feasibility Cut Enhancements

In this section, we describe the preliminary computational experiments for adding feasibility cuts to the Benders master problem. Our comparison takes into account three methods. The first is the use of rays obtained from the general-purpose software CPLEX 12.7.0 through its *getRay* function. The second is optimizing over the recession cone of DSP with an additional normalization constraint to bound the feasible space. Our final method is the use of cutset inequalities separated by efficient combinatorial algorithms.

To avoid the results being skewed by parameter fine-tuning of the Benders branch-and-cut approach, we implement these in the classic Benders decomposition algorithm where a complete enumeration tree is constructed to solve each master problem iteration. We next give a brief explanation of the two other methods used to obtain feasibility cuts.

Appendix A.1. Extreme Rays from the Simplex Algorithm

Benders feasibility cuts are defined by the rays of DSP. In practice, this is readily obtained from general purpose solvers. To properly understand the information provided by these solvers, we detail the definition and interpretation of an extreme ray of a linear program beginning with the definition of a recession direction.

A recession direction of a polyhedron $S \subset \mathbb{R}^n$ is a vector $d \in \mathbb{R}^n$ such that for any point $x \in S$ and for any $\lambda \geq 0$, $x + \lambda d \in S$. In other words, it is a direction in which you can take an arbitrarily large step and still be in S. An obvious necessary and sufficient condition for there to exist a non-trivial recession direction is that S be unbounded. The set of all recession directions forms a cone in $\mathbb{R}^n$ containing the origin. This cone will be referred to as the recession cone of S, $R_C(S)$.

A ray $r \in \mathbb{R}^n$ of a given linear program, in this case DSP, is a recession direction whose objective function evaluation is positive when maximizing or negative when minimizing. This implies that moving from any feasible point of DSP in the direction of r, the objective function value can be made arbitrarily large or small depending on the case. An extreme ray is a ray that cannot be expressed as a non-trivial convex combination of other rays of DSP.

In the case of IBM ILOG CPLEX, the ray provided by the *getRay* is the recession direction anchored at the last visited extreme point before detecting unboundedness (*http://www-01.ibm.com/support/docview.wss?uid=swg24042933*). In practice, there is no guarantee that such a ray is indeed extreme. We next present further details on another method used to obtain extreme rays to ensure feasibility.

Appendix A.2. Optimizing Over the Recession Cone of DSP

In this section, a different method for selecting feasibility cuts is proposed that, to the best of our knowledge, is novel not only in the context of network design, but also to Benders decomposition in general. The idea is to use an extreme ray that produces the highest objective function value increase (since DSP is a maximization problem) when one knows a priori that DSP is unbounded.

Recall from subsection Appendix A.1, that extreme rays are elements of the recession cone $R_C(DSP)$ that cannot be expressed as convex combinations of other points in

$R_C(DSP)$ and that have a positive value when evaluated in the objective function. Since $R_C(DSP)$ is unbounded, we must add a normalizing constraint that bounds it. The objective function of DSP can be maximized over this normalized recession cone $\bar{R_C}(DSP)$ and an extreme ray with the highest rate of increase for a given master problem solution $\bar{y}$ is obtained. Since we know a priori DSP is unbounded, then the optimal value of this problem is greater than 0.

The recession cone of DSP is characterized as follows:

$$R_C(DSP) = \{(\lambda, \mu) \in \mathbb{R}^{|N||K|} \times \mathbb{R}_+^{|A||K|} | \lambda_j^k - \lambda_i^k - \mu_{ij}^k \leq 0, \forall (i,j) \in A, \forall k \in K\}.$$

Note that because λ_i^k are unrestricted in sign $\forall i \in N, k \in K$, for a normalizing constraint to bound $R_C(DSP)$ we must rewrite the λ variables as $\lambda_i^k = \lambda_i^{k+} - \lambda_i^{k-}$ where $\lambda_i^{k+}, \lambda_i^{k-} \geq 0$ to use the normalization constraint $\sum_{k \in K} \sum_{i \in N} (\lambda_i^{k+} + \lambda_i^{k-}) + \sum_{k \in K} \sum_{(i,j) \in A} \mu_{ij}^k \leq 1$. The following optimization problem is used to determine the extreme rays that define the Benders feasibility cuts:

$$\begin{aligned}
\text{Rec(DSP)} \quad \max & \sum_{k \in K} (\lambda_{d(k)}^{k+} - \lambda_{d(k)}^{k-} - \lambda_{o(k)}^{k+} + \lambda_{o(k)}^{k-}) - \sum_{k \in K} \sum_{(i,j) \in A} \mu_{ij}^k \bar{y}_{ij} \\
\text{s.t.} & \lambda_j^{k+} - \lambda_j^{k-} - \lambda_i^{k+} + \lambda_i^{k-} - \mu_{ij}^k \leq 0 \quad \forall (i,j) \in A, \forall k \in K \qquad \text{(A.1)} \\
& \sum_{k \in K} \sum_{i \in N} (\lambda_i^{k+} + \lambda_i^{k-}) + \sum_{k \in K} \sum_{(i,j) \in A} \mu_{ij}^k \leq 1, \qquad \text{(A.2)}
\end{aligned}$$

where $\mu_{ij}^k \geq 0, \forall (i,j) \in A, \forall k \in K$ and $\lambda_i^{k+}, \lambda_i^{k-} \geq 0, \forall i \in N, \forall k \in K$.

$Rec(DSP)$ is large, in particular due to the duplicated λ variables. To circumvent this, we use the recession cone for each decomposed slave problem DSP_k and solve $|K|$ smaller LPs instead of solving the large LP Rec(DSP).

Appendix A.3. Feasibility cuts computational experiments

The classic Benders algorithm was coded in C using CPLEX 12.7.0 callable library and run on Intel Xeon processors at 3.10 GHz under a Windows environment. Table A.8 details the performance of each approach. ExtDSP refers to the method proposed in section Appendix A.1, Rec(DSP_k) refers to section Appendix A.2 and CutSet denotes the use of cutset inequalities. We show the number of feasibilty cuts (Feas Cuts), number of optimality cuts (Opt Cuts), Benders iterations (Iters), Iterations spent adding feasibility cuts (Feas Iter), CPU time and separation time. We note that more cutset inequalities than extreme rays are needed throughout the algorithm. However, the difference in time spent separating these is negligible as is the overall solution time to prove optimality. Based on these results, we decide in our Benders and cut implementation to use cutset inequalities to ensure feasibility.

Table A.8: Comparison of feasibility cut selection for Iterative Benders

Instance	$(\|N\|, \|A\|, \|K\|)$	Method	Feas Cuts	Opt Cuts	Iters	Feas Iter	CPU Time	Sep Time
		Rec(DSP_k)	49	156	11	6	0.16	0
c33.dat	(20,228,39)	CutSet	148	156	23	18	0.43	0.01
		ExtDSP	25	156	30	25	0.86	0.01
		Rec(DSP_k)	25	240	9	2	0.09	0
c35.dat	(20,230,40)	CutSet	68	240	15	8	0.16	0.01
		ExtDSP	20	240	27	20	0.67	0.02
		CutSet	78	240	15	8	0.17	0
c36.dat	(20,230,40)	Rec(DSP_k)	44	240	12	5	0.19	0.03
		ExtDSP	22	240	29	22	0.78	0.02
		Rec(DSP_k)	20	2400	14	1	372.62	0.06
c39.dat	(20,228,200)	CutSet	20	2400	14	1	404.95	0.09
		ExtDSP	1	2200	13	1	352.11	0.05
		CutSet	40	2000	14	3	47.42	0.07
c40.dat	(20,228,200)	Rec(DSP_k)	40	2000	14	3	51.57	0.07
		ExtDSP	3	2000	14	3	47.66	0.07
		CutSet	1	200	7	1	0.07	0.01
c41.dat	(20,288,40)	Rec(DSP_k)	1	200	7	1	0.08	0.01
		ExtDSP	1	200	7	1	0.09	0.02
		Rec(DSP_k)	37	320	13	4	1.36	0
c42.dat	(20,294,40)	CutSet	78	320	18	9	1.43	0.02
		ExtDSP	11	320	20	11	1.37	0
		Rec(DSP_k)	48	2400	9	2	1.28	0.16
c55.dat	(30,516,400)	CutSet	47	2800	10	2	1.29	0.2
		ExtDSP	6	2800	14	6	8	0.17
		Rec(DSP_k)	115	4000	16	5	74.69	0.31
c56.dat	(30,518,400)	CutSet	152	4400	18	6	76.88	0.32
		ExtDSP	8	4000	19	8	65.84	0.29
		CutSet	197	1000	23	12	4.57	0.11
c57.dat	(30,680,100)	Rec(DSP_k)	86	1100	19	7	4.74	0.08
		ExtDSP	21	1200	34	21	9.4	0.12
		CutSet	73	700	12	4	1.66	0.07
c59.dat	(30,687,100)	Rec(DSP_k)	21	700	10	2	1.89	0.06
		ExtDSP	3	700	11	3	2.45	0.07
		Rec(DSP_k)	122	800	14	5	15.83	0.06
c60.dat	(30,686,100)	CutSet	206	800	19	10	17.59	0.09
		ExtDSP	11	900	21	11	17.83	0.09
		CutSet	114	4400	16	4	152.24	0.41
c61.dat	(30,685,400)	Rec(DSP_k)	87	4800	16	3	237.23	0.44
		ExtDSP	6	4000	17	6	119.32	0.37
		Rec(DSP_k)	63	4400	15	3	10.4	0.39
c63.dat	(30,678,400)	CutSet	63	4400	15	3	12.08	0.42
		ExtDSP	3	4400	15	3	14.99	0.41

Appendix B. Core Point Selection Experiments

Within Benders branch-and-cut, the selection of the core point to be used plays a significant role in the speed of the algorithm. In the following table, we compare two approaches for this. The first, Benders NF, is the core point selection strategy of updating it at every separation for Benders cuts as the midpoint between the current master problem solution and the previous core point. On the other hand, Benders FC represents

maintaining the core point fixed at the point partially defined by the union of the arcs that comprise the shortest path.

Table B.9 shows the importance of proper core point selection, obtaining a speed-up of an order of magnitude when using the appropriate strategy.

Table B.9: Comparison varying core point strategy

		Benders NF			Benders FC		
Class	Description	Nodes	Benders Cuts	CPU time	Nodes	Benders Cuts	CPU time
	20,230,40	3.33	660	0.1	0	19.67	0.61
	20,300,40	1.75	751	0.11	0.75	38.25	0.71
I	30,520,100	6438.25	5890.5	660.06	2978.5	2896.25	222.08
	30,700,100	154.5	3184	11.79	114.5	934.25	9.22
	20,230,200	532.5	7796.25	56.34	273.5	4643.25	15
	20,300,200	372	7144.75	36.34	151.5	3739.25	10.73
II	30,520,400	55	7198	7.31	26.5	2203.25	13.06
	30,700,400	8557.25	19784.25	2832.44	3841.75	4686.25	237.53
	Testbed Average	2014.32	6551.09	450.56	923.38	2395.05	63.62