UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

RESOLUCIÓN DE PROBLEMAS DE DISEÑO DE REDES MEDIANTE DUAL-ASCENT PARA APLICACIONES INDUSTRIALES

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN GESTIÓN DE OPERACIONES

SEBASTIÁN ANDRÉS RIVAS SÁENZ

PROFESOR GUÍA:
RAFAEL EPSTEIN NEHUMHAUSER

MIEMBROS DE LA COMISIÓN:
MARCELO OLIVARES ACUÑA
JOSE CORREA HAEUSSLER
PABLO ANDRES REY

Este trabajo ha sido parcialmente financiado por:
Comisión Nacional de Investigación Científica y Tecnológica (CONICYT)

SANTIAGO DE CHILE
2016

Resumen

En este trabajo se desarrolla un nuevo enfoque para resolver el problema de diseño de redes no capacitadas con fuente única en base a la combinación de métodos desarrollados en estudios previos. Para este tipo de problemas, la formulación multicommodity que desagrega las demandas ha sido utilizada extensamente y se ha probado que se obtienen mejores resultados que con la formulación de flujo en redes clásica al comparar sus relajaciones lineales. En este trabajo se muestra que dicha formulación puede mejorar aún más al duplicar y dirigir arcos no-dirigidos. Con este concepto, se desarrolla un método de ascenso dual específico para el problema de diseño con fuente única que entrega cotas inferiores de buena calidad. Dentro de este método se propone un esquema de clasificación de commodities que permite una representación reducida del problema y que entrega mejores cotas inferiores en las instancias testeadas. Adicionalmente, este método también entrega una subred de tamaño reducido que se utiliza para encontrar soluciones primales factibles. Se muestra, que en este sentido, el método de ascenso dual es una excelente herramienta de selección de arcos en términos del potencial que tiene la subred de encontrar soluciones primales de buena calidad. Para obtener la solución primal, se utiliza la formulación multicommodity original o un esquema de generación de filas dependiendo del tamaño de la instancia. Se testean los distintos enfoques en instancias de distintos tamaños de redes en forma de grilla generadas aleatoriamente variando sus parámetros y su relación de costos fijos a costos de flujo, testeando instancias que en su equivalente de formulación multicommodity llegan a más de 16 millones de variables.

Abstract

We develop a new approach to solve the single source uncapacitated network design problem by combining different methods developed in previous studies. For these types of problems, the multicommodity formulation that disaggregates demands has been extensively used and has proved to be tighter in the linear relaxation than the classic network flow formulation. We show that such formulation can further improve by duplicating and directing non-directed edges. With this concept we develop a specific dual-ascent procedure for the single source problem that is capable of giving good quality lower bounds to the problem. We propose a commodity sorting scheme to solve this dual-ascent procedure that allows a smaller representation of the problem and delivers better lower bounds. Additionally, this procedure also delivers a reduced size sub-network that we use to find feasible primal solutions. We show that, in this sense, the dual-ascent procedure is an excellent arc-selecting tool in terms of the capacity of the sub-network of obtaining good quality primal solutions. Depending on the size of the instance, the original directed multicommodity formulation or a row generation scheme is used to find the upper bounds to the problem. We test the different approaches on random instances of gridded networks of different sizes (up to an equivalent of approximately 16 million variables in the directed multicommodity formulation) varying their parameters and their fixed to flow cost relation.

Agradecimientos

Al profesor Rafael Epstein por la confianza que depositó en mí y por sus consejos en el trabajo y en la vida.

A los profesores miembros de la comisión: Pablo Rey por su minuciosa revisión y asertivos comentarios; a Marcelo Olivares por la oportunidad de continuar esta investigación; a José Correa por su disposión a participar de este trabajo y el conocimiento que me entregó durante la carrera.

A Linda Valdés por su gestión en la recta final.

A la Comisión Nacional de Investigación Científica y Tecnológica por financiar parcialmente este trabajo.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Network design problems typically involve finding the best way to transport a given commodity from an origin to a destination. The problem is generally modeled through a network consisting of a set of nodes and a set of arcs that represent the studied space. Some nodes could represent a supply for a determined commodity and others could represent demands. With this model, the objective is to choose the correct subset of nodes and arcs that are able to satisfy the demands while also minimizing the transportation costs. Specifically in the design problems, the transportation costs not only consider the flow cost for moving the commodity, but also a fixed cost usually related to the construction of a road or means for transport. This modeling strategy has proved very useful in its application to problems of Electricity, Telecommunications, Transportation, Logistics and Planning among others. A good survey of applications of this problem may be found in [10].

In terms of applied mathematics, there already exist several approaches to solve the previously described problem. Basically, it seeks to optimize a value subject only to a finite number of alternatives and therefore it can be modeled as a mixed integer programming problem. Such problems already have several exact algorithms to find the optimal solution. However, in real industry applications, the number of variables and constraints grow in such a way that traditional methods become inefficient, as they take impractically large amounts of time to obtain solutions. In this context, the scope of study focuses in finding good alternative algorithms or heuristics that can ensure that the execution times do not grow significantly with the growth of the network under study even in expense of getting an exact optimal solution. This study continues the work originally proposed in [12] and the main structure for the implementation of the studied methods are strongly based on the work in [3], where the use of the Dual-Ascent strategy for the the network desing problem was first presented.

1.1 Network Design Problems

The concept of Network Design Problems is a general classification for a wide set of problems which differ from each other according to specific parameters or constraints. All of them have in common the usual network model, which consists of a set of nodes and a set of arcs or edges. Nodes usually represent origin, destination or transit points for determined inputs or commodities. Each arc or edge can have minimum or maximum flow capacity, a fixed cost that has to be paid if the arc is used in the final solution, and a variable cost per unit of flow. An accurate study of classical network flow problems and a survey of both algorithms and data structures for their implementation can be found in [1]. As discussed in such study, various special cases can be obtained either by limiting the network structure, fixing costs and/or restraining the amount of supply or demand nodes. In this regard, the network problems can be classified according to the following features:

- *Capacity*: Arcs with or without minimum or maximum capacity constraints.
- *Type of Network*: Directed arcs or non-directed edges.
- *Offer/Demand Structure*: Multiple origins and destinations or single source.
- *Cost Structure*: Fixed costs, linear flow costs, non-linear costs, among others.
- *Commodities*: Multiple or single commodity flows.
- *Network Structure*: Complete graphs, bipartite graphs, multiple layers, among others.

Using these features, the specific problem that we study can be classified as an uncapacitated network design problem in non-directed graphs with a single commodity source that considers both fixed costs and linear flow costs, or in short *SS-UND* (single source uncapacitated network design). In [10] it is shown that with some slight modifications, this particular problem is actually a generalization of several known and deeply studied classical optimization problems. This list considers the Travelling Salesman Problem, Facility Location Problem, Minimum Cost Flow, Minimum Spanning Tree and the Steiner Tree Problem, among others. Many of these problems are difficult to solve in a computational complexity sense (NP-Hard), so as a generalization of those problems, we can conclude that the problem under study is also NP-Hard. Furthermore, regarding the mentioned problems, the network design problem can be interpreted as a combination of the Minimum Cost Flow and the Steiner Tree Problem. On one hand, in the Steiner Tree Problem, the objective is to find the most efficient way of connecting a subset of nodes by paying the fixed costs of including the corresponding edges. In the Network Design Problem, this is the minimum requirement that enables to get flow to the demand nodes. On the other hand, in the Minimum Costs Flow Problem, demand nodes are already connected and the requirement is to find the better way to route flow to achieve the minimum cost. In network design problems both of these problems must be solved simultaneously. These two components provide a trade-off between flow and fixed costs (or fixed and operational costs). On one side, if more arcs are included, arcs with lower flow costs might be available at the expense of higher fixed costs. On the other side, by selecting fewer arcs in the design and thus, reducing the fixed costs, the flow costs might eventually increase.

1.2 Objectives

The general objective of this work is to combine algorithms and heuristics to develop strategies to efficiently solve the Single Source Uncapacitated Network Design Problem. To accomplish such objective, the following specific objectives where defined.

- Study existing formulations and solution approaches for the problem under study.
- Explore existing algorithms and heuristics to solve similar problems.
- Analyze alternative strategies or adaptations of studied methods to improve their efficiency in terms of CPU memory usage and execution times.
- Program algorithms and heuristic and create means of measuring their performance in a testing environment

Chapter 2

Framework

2.1 Mathematical Model

In this section, different formulations for the SS-UND problem are presented and discussed in order to identify their advantages and disadvantages. Considering this discussion, we then propose a strengthening adaptation for one of the formulations that we implement, improving its quality in terms of their linear relaxation tightness.

Independently of the specific formulation, an instance of the SS-UND problem is constituted by a set of nodes N, a set of non-directed edges E, a subset of nodes with positive demand T, a single source $s \in N$, and the following parameters:

d_k :	Demand of node $k \in T$
D :	Total demand ($D = \sum d_k$)
F_e :	Fixed cost for including edge $e \in E$ in the network
C_e :	Cost per unit of flow through edge $e \in E$

2.1.1 Network Flow Formulation

A classic formulation for this problem is the Network Flow Formulation (NF), which is based on the Minimum Cost Flow model. As in the mentioned model, the NF formulation contains the continuous variables x_{ij} that represents a directed flow on a specific direction through the non directed edge $e = \{i, j\}$ with $i, j \in N$. Additionally, a set of binary variables y_e are incorporated to the model to represent the inclusion or exclusion of a particular edge $e \in E$ in the design. As usual, for $S \subset N$ we use $\delta(S)$ to denote the set of edges in the cut defined by S, namely, $\delta(S) = \{\{i, j\} \in E : i \in S, j \in N \backslash S\}$. The model has the following form.

Problem NF

$$\min \sum_{e=\{i,j\}\in E} (F_e y_e + c_e(x_{ij} + x_{ji}))$$

subject to

$$\begin{aligned} x_{ij} &\le D \cdot y_e \\ x_{ji} &\le D \cdot y_e \end{aligned} \qquad \forall e = \{i,j\} \in E \tag{2.1}$$

$$\sum_{\{i,j\}\in\delta(i)} (x_{ij} - x_{ji}) = \begin{cases} D & \text{if } i = s \\ -d_i & \text{if } i \in T, \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in N \tag{2.2}$$

$$x_{ij}, x_{ji} \ge 0 \qquad \forall \{i,j\} \in E \tag{2.3}$$

$$y_e \in \{0,1\} \qquad \forall e = \{i,j\} \in E \tag{2.4}$$

Constraints (2.2), imposed over every node, are the usual flow conservation restrictions. Particularly, all flow must start at node s as a consequence of the single source structure. Constraints (2.1), imposed over every edge, force a design variable to be included if its corresponding flow variable is non zero (on either direction), and in effect, pay its fixed cost. This is a common constraint to relate binary and continuous variables and the use of a sufficiently large constant is required. In this case, the total sum of demand will be enough considering that no flow on any edge will ever be greater than that value on an optimal solution (non negative costs imply that an optimal basic solution without flow cycles does exist).

As shown in [15], considering a linear relaxation to solve this problem, this formulation is weak both in theory and practice. Generally, the linear relaxation of NF, which we call NF_L, is loose in the optimal integer variables and in the value of the objective function.

Theoretically, the formulation is weak for the following reason. As already mentioned, the optimal solution does not present flow cycles, so each edge will at least present a null flow on one of the directions (or both). That is $x_{ij} = 0$ or $x_{ji} = 0$ for all $e \in E$. Therefore, constraints (2.1) can be replaced by:

$$x_{ij} + x_{ji} \le D y_e \qquad \forall e = \{i,j\} \in E \tag{2.5}$$

Variables y_e always tend to be as smaller as possible since it has a positive coefficient in the objective function, but constraints (2.1) force these variables to take the value of 1 on certain edges. When relaxing integrality, these are the only constraints restraining the design variables to be zero, and thus they became active for all edges (even on those where there is no flow at all). This implies that y_e can be replaced by $(x_{ij} + x_{ji})/D$, and therefore obtaining a linear model which optimal basic solution is a tree of shortests paths from node s to every demand node using the modified costs equal to $c_{\{i,j\}} + F_{\{i,j\}}/D$. This clearly eliminates the coupling relationship between nodes (the Steiner Tree component of the problem). Moreover, D is an intrinsic parameter to a specific instance and it can even take any other arbitrary value sufficiently large. These observations show that the linear relaxation of the Network Flow formulation will not provide a good approximation of the solution.

2.1.2 Multicommodity Formulation

A disaggregate demand model is proposed in [14] mainly to address the problems associated to the presence of an arbitrary constant in the forcing constraints (2.1). This model, known as the Multicommodity Formulation (MC), has been discussed in numerous articles [15, 8] and has proven to give tighter gaps than the previous formulation.

In this formulation, each demand represents a different commodity that can flow through any edge. Let K denote the set of commodities that have to flow each node $i \in T$, and let $D(k)$ denote the destination node in T associated to commodity $k \in K$. Considering that any edge may present flows of different commodities, we replace variable x_{ij} with a variable x_{ij}^k that represents the fraction of commodity k that flows from node i to node j through the edge $\{i, j\}$. By recovering the cost structure of the previous model, naturally the flow cost associated to this variable is equal to $c_{\{i,j\}} \cdot d_{D(k)}$. We abbreviate such cost with $c_{\{i,j\}}^k$. In accordance with this new notation, the Multicommodity formulation is the following.

Problem MC

$$\min \sum_{e=\{i,j\}\in E} \left(F_e y_e + \sum_{k\in K} c_e^k (x_{ij}^k + x_{ji}^k) \right)$$

subject to

$$\begin{aligned} x_{ij}^k &\leq y_e \\ x_{ji}^k &\leq y_e \end{aligned} \qquad \forall e = \{i,j\} \in E, \forall k \in K \tag{2.6}$$

$$\sum_{\{i,j\}\in\delta(i)} (x_{ij}^k - x_{ji}^k) = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i \in T, \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in N, \forall k \in K \tag{2.7}$$

$$x_{ij}^k, x_{ji}^k \geq 0 \qquad \forall \{i,j\} \in E, \forall k \in K \tag{2.8}$$

$$y_e \in \{0,1\} \qquad \forall e = \{i,j\} \in E \tag{2.9}$$

As mentioned before, the strategy of disaggregating the demands yields the benefit of removing the external arbitrary constant from the forcing constraint (2.1). We now show how this change improves the quality of the linear relaxation solution in comparison to the previous formulation.

When solving the linear relaxation of NF, the design variables y_e take a value equal to the fraction of the total demand that flows throw the edge $e = \{i, j\}$. Following that same argument, in this case we can state that constraints (2.6) can be replaced by:

$$x_{ij}^k + x_{ji}^k \leq y_e \qquad \forall e = \{i,j\}, k \in K \tag{2.10}$$

When solving the linear relaxation of MC (MC_L) these constraints (2.10) are active, and therefore the design variables are equal to $\max_{k\in K}\{x_{ij}^k + x_{ji}^k\}$, the maximum fraction of the demands of commodities that flow throw edge $e = \{i,j\}$. This simple observation shows that the optimal value of MC_L is greater than or equal to NF_L.

Let ($\hat{x}$,$\hat{y}$) be an optimal solution of MC_L. A feasible solution ($\widetilde{x}$, $\widetilde{y}$) of NF_L can be constructed as follows:

$$\widetilde{x}_{ij} = \sum_{k\in K} \hat{x}_{ij}^k d_{D(k)} \qquad \widetilde{x}_{ji} = \sum_{k\in K} \hat{x}_{ji}^k d_{D(k)} \qquad \widetilde{y}_e = \frac{\widetilde{x}_{ij} + \widetilde{x}_{ji}}{D} \qquad \forall e = \{i,j\} \in E \tag{2.11}$$

The flow costs component of both solutions are exactly the same, as we show with a simple rearrangement of the flow cost expression:

$$\sum_{e\in E}\sum_{k\in K} c_e^k(\hat{x}_{ij}^k + \hat{x}_{ji}^k) = \sum_{e\in E}\sum_{k\in K} c_e \cdot d_{D(k)}(\hat{x}_{ij}^k + \hat{x}_{ji}^k) = \sum_{e\in E} c_e \sum_{k\in K} d_{D(k)}(\hat{x}_{ij}^k + \hat{x}_{ji}^k) = \sum_{e\in E} c_e(\widetilde{x}_{ij} + \widetilde{x}_{ji})$$

However, the design costs component is not necessarily the same. In the case of the MC_L formulation, as mentioned above, the design variables $\hat{y}_e$ are equal to $\max_{k\in K}\{\hat{x}_{ij}^k + \hat{x}_{ji}^k\}$ and for the NF_L formulation, the design variables $\widetilde{y}_e$ are equal to $\frac{\sum_k (\hat{x}_{ij}^k + \hat{x}_{ji}^k)\cdot d_{D(k)}}{D}$. Given such expressions, clearly $\hat{y}_e \geq \widetilde{y}_e$. This shows the advantage of the MC formulation with respect to the NF formulation from a point of view of the quality of optimal solution of the linear relaxation.

Despite the benefits of this formulation, it also generates a new problem that limits its effectiveness. By disaggregating demands, $O(|T|\cdot|E|)$ additional variables and $O(|T|\cdot(|E| + |N|))$ new constraints are added to the problem. Even when relaxing integrality, the size of the problem soon becomes unmanageable by traditional linear programming tools. This is a particularly impeding limitation when considering industrial-size instances. Just to give an example, relatively small instances of 300 nodes, 200 destination nodes, in networks with 20% of edge density have around four million variables and the same amount of constraints.

2.1.3 Dicut Formulation

Due to the problems of the Multicommodity Formulation, many researches have been focused on finding facets and valid inequalities for NF that could enhance the linear relaxation.

In this line, [17] proposes one type of these inequalities that have been effective in reducing the linear gap. In [15], a family of inequalities is proposed, which describe the projection of the MC formulation variables over the space of variables of the NF formulation and thus, making their linear relaxation equivalent. However, the number of these inequalities grows exponentially with the problem size, therefore a row generation scheme is proposed. The problem of finding efficient algorithms to solve the separation problem (identifying the violated cuts to be able to add them individually) is still open. In [13] a branch and cut algorithm is developed to solve the SS-UND problem and various types of cuts are introduced. For some of these inequalities, called *single-cuts*, the separation problem can be efficiently solved. For the rest of the cuts, a heuristic is used to identify violated inequalities. The simple cuts are equivalent to the cutting planes used by [6], [9] and [4] for the Steiner Tree Problem (STP). We propose the following directed formulation for the SS-UND by using these simple cuts as an alternative to the previous formulations.

Problem $DICUT$

$$\min \sum_{(i,j)\in A} F_{ij}y_{ij} + \sum_{(i,j)\in A} c_{ij}x_{ij}$$

subject to

$$\sum_{(i,j)\in\delta_+(S)} y_{ij} \geq 1 \qquad \forall S \subset V, (V \setminus S)\cap T \neq \phi, s \in S \tag{2.12}$$

$$x_{ij} \leq D \cdot y_{ij} \qquad \forall (i,j) \in A \tag{2.13}$$

$$\sum_{j\in N}(x_{ij} - x_{ji}) = \begin{cases} D & \text{if } i = s \\ -d_i & \text{if } i \in T, \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in N \tag{2.14}$$

$$x_{ij}, x_{ji} \geq 0 \qquad \forall (i,j) \in A \tag{2.15}$$

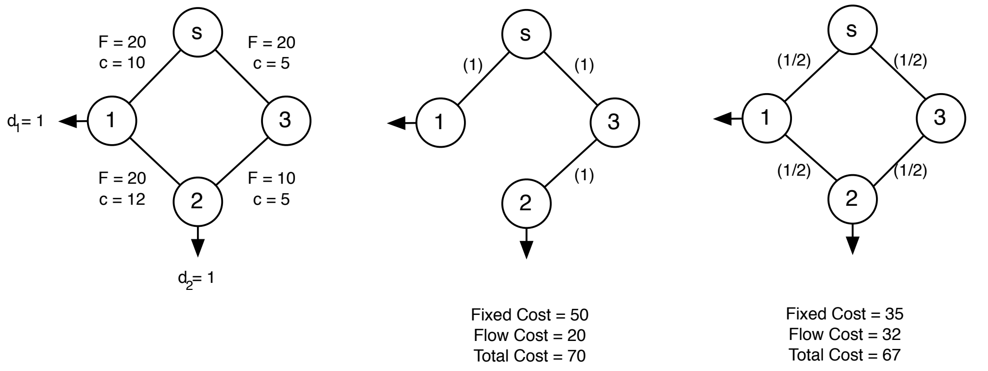
$$y_{ij} \in [0,1] \qquad \forall (i,j) \in A \tag{2.16}$$

Just as the di-cut formulation for the STP proposed in [2] and also used in [4], [5] and [8], the separation problem of constraints (2.12) is reduced to a min cut problem (or max flow). First, each design variable value is assigned as an arc capacity. Then, if the min cut between the source and any destination node has a capacity less than 1, then such cut is added to the constraints in (2.12). Unlike the STP, this problem also has a flow cost component, therefore such variables cannot be eliminated. This makes the Dicut formulation for this problem non equivalent to MC, and thus small gaps cannot guaranteed. Further in this subject will be presented in the following sections. We could add the constraints proposed in [15], but we would lose efficiency in the separation problem. We rather opt to add only the simple-cuts and find the violated ones in an exact (non-heuristic) way.

2.2 Strengthening the Model

Although the MC formulation is the best alternative in terms of quality of its linear relaxation solution, in certain networks, it can still present significant gaps with respect to the integer solution. In this section, we show what causes this problem and propose means to strengthen the formulation to improve the gap.

Figure 2.1 illustrates an instance where the MC_L formulation fails to obtain the optimal integer solution.



(a) Undirected graph (b) Optimal integer solution (c) Optimal relaxed solution

Figure 2.1: Cycled solution on MC linear relaxation

Figure 2.1(b) shows the integer optimal solution according to the parameters of the instance shown in 2.1(a). The values in parenthesis denote the value of the design variables y_e associated to each edge. The total cost of the solution is 70, 50 for the fixed cost and 20 for the flow cost. Figure 2.1(c) shows the optimal solution of MC_L. In this case, the design variables y_e associated to each edge take a value of 1/2. Considering constraint (2.6), each edge only allows a maximum flow of 1/2. To satisfy the full unit of the demand of node 1, half of the flow goes through path (s,1) and the other half goes through path (s,3,2,1). The cost for commodity 1 is $(10) \cdot (1/2) + (5 + 5 + 12) \cdot (1/2)$, which results in 16. Similarly, to satisfy the full unit of the demand of node 2, half the flow goes through path (s,1,2) and the other half goes through path (s,3,4). The cost for commodity 2 is $(10 + 12) \cdot (1/2) + (5 + 5) \cdot (1/2)$, which also results in 16, giving a total flow cost of 32. Then, the total cost of the solution is 67, 32 for the flow cost and 35 for the fixed cost. This example shows that MC_L can still present a high optimal gap.

In this particular instance, the problem clearly is the fractional values of the optimal design variables. However, what causes this is the cyclic structure of the solution, allowing the existence of two possible paths from the source to each destination, and hence a lower cost solution. Another important aspect of this example is that edge $\{1, 2\}$ presents flow in opposite directions. For commodity 1, flow goes from node 2 to node 1, and for commodity 2, flow goes from node 1 to node 2. If this could be prevented, then the cyclic structure cannot grant any benefit over the tree structure of the integral solution, and thus the original optimal solution could be recovered with the linear relaxation model.

We previously discussed three formulations for the SS-UND. The NF and MC formulations are both non-directed in their edge design variables (motivated by the non-directed nature of the original problem). However, the Dicut formulation presents directed arc variables. We now show that the MC formulation can improve by incorporating such quality. This is accomplished by replacing each edge $e = \{i, j\}$ by two arcs (i, j) and (j, i) with equal fixed and flow cost. Intuitively, this modification will incorporate a cost penalization of flows on opposite directions. If flows occur on both directions on an edge, now both design variables (i, j) and (j, i) must be built and both will have to pay the associated fixed cost. For further discussions, we use A to denote the set of directed arcs instead of E that denotes the set of non-directed edges. Nevertheless, we sometimes maintain the use of edges to explicitly show that some constraints or variables will appear in both directions. In such cases, the sets E and A denote both the same network, either in edge or arc representation accordingly.

On the rest of this section we generalize the observations made on the previous example. We show that in fact, the integer optimal solution will never present cycled nor opposite direction flows, which can occur on the relaxed solution. Following the previous, we present the formulation over the directed network and show that it is equivalent to the non-directed version in the discrete problem. We base the analysis upon three basic characteristics of the SS-UND problem: (i) the problem is uncapacitated; (ii) there is a single source for all commodities, (iii) Each arc has equal flow costs for every commodity (as in the original problem, where in fact, there was a single commodity with multiple demand sinks.)

Proposition 1 There is always an optimal solution for the SS-UND problem that is a tree.

Proof. Given a feasible solution y for the design variables (that is, a subset $\tilde{E} \subset E$ with $y_e = 1 \; \forall e \in \tilde{E}$ such that s is connected to every node $i \in T$), we can assign optimal values for the flow variables x by solving a Shortest Path Problem from s to each node in T. Given that x_e cannot be positive for any e where $y_e < 1$, then let $y_e = 0 \; \forall e \notin \tilde{E}$. Considering that the cost per unit of flow is equal for every commodity, then the shortest paths do not depend on the demand of each commodity d_k. Also, since there is a single source, this problem is solved by calculating the shortest path tree from s to every other node in N and then iteratively eliminating the *leaves* of the tree that do not end in a node in T. Then, if each arc without flow is eliminated from y ($y_{ij} = 0$ if $x_{ij} = 0$), we obtain a tree as the final discrete solution for the problem. Note that this final step could only cause an improvement in the total solution cost. In conclusion, with any feasible solution, another tree-structured feasible solution with equal or smaller cost can be constructed, and hence the SS-UND always allows a tree-structured optimal solution. Note that in cases with edges with $F_e = 0$ or $c_e = 0$, multiple optimal solutions could exist, however using the above procedure, a tree-structured optimal solution can always be found. □

Proposition 2 There is always at least one optimal solution (y, x), in which there are no arcs with flow in opposite directions, i.e. $(x^k_{ij} = 1)$ with $k \in K \Rightarrow (x^h_{ji} = 0) \; \forall h \in K$.

Proof. Let y^* be the optimal design variables. As previously discussed, given the design variables, the optimal choice for the flow variables x^* does not depend on the individual commodity demands d_k. Therefore, for simplicity and without loss of generality, let us assume $d_k = 1, \forall k \in K$, and in consequence $c^k_{ij} = c_{ij}, \forall k \in K, \forall \{i,j\} \in A$.

Suppose there exists an optimal solution with an edge $e = \{i^*, j^*\}$, such that $x^k_{i^*j^*} = 1, k \in K$ and for other certain commodity $h \in K$ there also exists flow in the opposite direction, ie. $x^h_{j^*i^*} = 1$

Since the optimal values for the flow variables x are solving a Shortest Path Problem, let $l_{i,j}$ denote the length (or cost) of the shortest path from node i to node j (path $i - j$). Then, considering $x^k_{i^*j^*} = 1$, the path from s to $D(k)$ must include arc (i^*, j^*) , so the length from s to j^* is:

$$l_{s,j^*} = l_{s,i^*} + c_{i^*j^*}$$

On one hand, if $c_{i^*j^*}$ is positive, then we have $l_{s,j^*} > l_{s,i^*}$.

Using the same argument as before, given that $x^h_{j^*i^*} = 1$, arc (j^*, i^*) must be in a path $s - D(h)$, and hence we can state:

$$l_{s,i^*} = l_{s,j^*} + c_{j^*i^*}$$

Therefore, we can also conclude that $l_{s,i^*} > l_{s,j^*}$, which is a contradiction that discards the existance of flow in opposite directions.

On the other hand, if $c_{i^*j^*} = 0$, then $l_{s,i^*} = l_{s,j^*}$. In this case, there will be multiple optimal possible optimal solutions. One of these solutions is to send both commodities k and h together from s to i^*. Commodity k can flow through arc (i^*, j^*) and from j^* to $D(k)$. Commodity h instead can flow from i^* to $D(h)$ without flowing through arc (j^*, i^*). In such optimal solution, we have $x^h_{j^*i^*} = 0$.

□

Proposition 2 enables improvements which, although redundant in the discrete problem, can reduce the gap in the linear relaxation. One way to strengthen the MC formulation is to introduce the discussed result as a linear constraint:

$$x^k_{ij} + x^h_{ji} \leq y_e \qquad \forall h, k \in K, \forall e = \{i,j\} \in E \tag{2.17}$$

This constraint could replace the forcing constraints (2.6) and will contribute to avoid flow in opposite directions. However, doing so will add $O(K^2 \cdot E)$ additional constraints to the problem, and thus contributing even further with the unmanageable size of real instances. For further discussion, we refer to this new formulation as the Multicommodity Extended formulation (MCE).

As already mentioned, another way to avoid flow in opposite directions is to direct the design variables by replacing each edge by two arcs. We refer to this new formulation as the

Multicommodity Directed Formulation (MCD). Several articles for similar combinatorial problems have reported that this modification tightens the linear relaxation gap. For example, in [11] the directed formlation for the Minimum Spanning Tree and the Steiner Tree problem is discussed.

In the integer problem, both models are equivalent, in account of reaching optimality only one arc in a determined direction is built, recovering the non-directed optimal solution. Yet, the great advantage of MCD is the benefits of its linear relaxation, which we call MCD_L. First, the linear model MCD_L has the same number of constraints as MC_L and only doubles the number of design variables. Second, as we will prove, its linear relaxation solution is identical to the linear relaxation of the Extended Multicommodity Formulation MCE_L without the additional exponential number of constraints.

Consider the following formulation for MCD:

Problem MCD

$$\min \sum_{e=\{i,j\}\in E} \left(F_e(y_{ij} + y_{ji}) + \sum_{k\in K} c_e^k (x_{ij}^k + x_{ji}^k) \right)$$

subject to

$$\begin{aligned} x_{ij}^k &\le y_{ij} \\ x_{ji}^k &\le y_{ji} \end{aligned} \qquad \forall e = \{i,j\} \in E, \forall k \in K \tag{2.18}$$

$$y_{ij} + y_{ji} \le 1 \qquad \forall \{i,j\} \in E \tag{2.19}$$

$$\sum_{\{i,j\}\in\delta(i)} (x_{ij}^k - x_{ji}^k) = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i \in T, \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in N, \forall k \in K \tag{2.20}$$

$$x_{ij}^k, x_{ji}^k \ge 0 \qquad \forall \{i,j\} \in E, \forall k \in K \tag{2.21}$$

$$y_{ij}, y_{ji} \in \{0,1\} \qquad \forall e = \{i,j\} \in E \tag{2.22}$$

This formulation forces to declare on which direction each edge is used before it is constructed, causing that twice the costs is paid if it is used on both directions. Nevertheless, we also add constraint (2.19) to avoid such situation in the integer problem. Based on this, the optimal solution for MCD will have flow only on one direction in each arc, then using the result in proposition 2 we can state that this solution is also optimal on formulation MC (In general terms, the rest of the problem is the same). With this we have shown that the directed formulation MCD is valid for the SS-UND problem. The following proposition shows the benefits of the linear relaxation MCD_L.

Proposition 3 The MCD_L problem is equivalent to the MCE_L problem in terms of optimal values of objective functions. Also, an optimal solution of MCD_L is optimal in MCE_L and an optimal solution in MCE_L can be projected to an optimal solution of MCD_L.

Proof. Let $(\hat{y}, \hat{x})$ be an optimal solution of MCD_L. Define $(\tilde{y}, \tilde{x})$ as vectors for the variables of MCE_L such that $\tilde{y}_e = \hat{y}_{ij} + \hat{y}_{ji}$, $\forall e = \{i,j\} \in E$ and $\tilde{x} = \hat{x}$. We can see that $(\tilde{y}, \tilde{x})$ is a feasible solution of MCE_L and that it has the same cost as $(\hat{y}, \hat{x})$

Now Let $(\tilde{y}, \tilde{x})$ be an optimal solution of MCE_L. Let $(\hat{y}, \hat{x})$ be vectors for the variables of MCD that take the following values: $\hat{y}_{ij} = \max_k\{\tilde{x}^k_{ij}\}$, $\hat{y}_{ji} = \max_k\{\tilde{x}^k_{ji}\}$, for all $e = \{i,j\} \in E$ and $\hat{x} = \tilde{x}$.

Given that for any commodity pair (k, h) it holds that $\tilde{x}^k_{ij} + \tilde{x}^h_{ji} \leq \tilde{y}_e$, we can directly conclude that $\hat{x}^k_{ij} + \hat{x}^h_{ji} \leq 1$. Also, by construction $(\hat{y}, \hat{x})$ satisfies constraint (2.18), therefore it is a feasible solution in MCD_L. Since $(\tilde{y}, \tilde{x})$ is optimal in MCE_L, as discussed after presenting the MC formulation, we have $\tilde{y}_e = \max_k\{\tilde{x}^k_{ij}\} + \max_k\{\tilde{x}^k_{ji}\}$ for all $e = \{i,j\} \in E$. This implies that $\tilde{y}_e = \hat{y}_{ij} + \hat{y}_{ji}$. Therefore, both solutions have the same cost. This shows that a feasible solution in MCD_L can be lifted from an optimal solution in MCE_L and that it will have an equal cost. $\square$

Figure 2.2 illustrates the same instance previously described in figure 2.1 and how this time, the integer optimal solution can be obtained through MCD_L formulation.

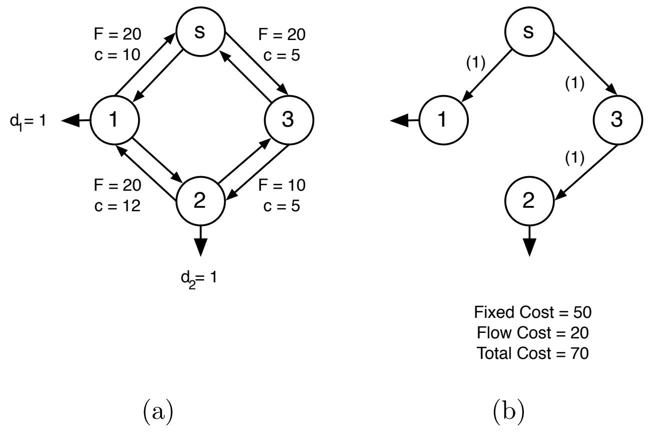


Figure 2.2: Cycle elimination through directed formulation

Each non-directed edge is replaced by two directed arcs with the same flow and fixed cost as the original edge, resulting in the instance of figure 2.2(a). The solution of the directed linear relaxation, shown in figure 2.2(b), does not have cycles nor opposite directed flows and reaches the same cost as the integer optimal solution.

Constraint (2.19) is a key element to understand that MCD will not have a higher cost than the non-directed formulation. However, we now prove that it can be removed from the formulation without affecting the optimal solution. Essentially, the cost penalization of paying twice the fixed cost to allow flow in opposite directions is enough to avoid it. This result is of great use for the design of the dual ascent algorithm for the next section.

We first define the concept of *tight optimal solution* for MCD_L. If $F_{i,j}$ is strictly positive, it is clear that:

$$y_{ij} = \max_k\{x^k_{ij}\} \qquad \forall\{i,j\} \in E, k \in K \qquad (2.23)$$

However, if $F_{i,j}$ is equal to zero, then y_{ij} can take any value between $\max_k\{x_{ij}^k\}$ and 1 and the solution will still be optimal. We denote as tight optimal solution to those solutions that satisfy (2.23) for all edges.

Let $MCDR$ be the formulation without considering constraint (2.19), which we call the Reduced Multicommodity Directed Formulation, and $MCDR_L$ its respective linear relaxation.

Theorem 1 Problems MCD_L and $MCDR_L$ have the same set of tight optimal solutions.

Proof. Let (y, x) be a tight optimal solution for $MCDR_L$. We will prove that $y_{ij} + y_{ji} \leq 1$ for all $e = \{i, j\} \in E$ (i.e. constraint (2.19)).

Constraint (2.19) is clearly satisfied if, for a determined edge, there is flow on a single direction or no flow at all, hence we will only analyze the case where there is flow on both directions. Let $\{i^*, j^*\}$ be an edge with flow on both directions. Let $p = \text{argmax}_k\{x_{i^*j^*}^k\}$ and $q = \text{argmax}_k\{x_{j^*i^*}^k\}$. Given that (y, x) is a tight optimal solution, then:

$$y_{i^*j^*} = x_{i^*j^*}^p \text{ and } y_{j^*i^*} = x_{j^*i^*}^q \tag{2.24}$$

Also, for a specific commodity, no node will receive a total flow greater than the demand for such commodity. Particularly, for commodity q and node i^*, it holds that:

$$\sum_{t\in N} x_{ti^*}^q = \sum_{t\in N\setminus\{j^*\}} x_{ti^*}^q + x_{j^*i^*}^q \leq 1 \tag{2.25}$$

Considering that $x_{i^*j^*}^p > 0$ and given that (y, x) is optimal, flow for commodity p in the opposite direction must be zero; i.e. $x_{j^*i^*}^p = 0$. Also, flow p that enters node i^* must, at least, be equal to the flow that is already leaving this node; $x_{i^*j^*}^p$. Therefore, we can state that:

$$\sum_{t\in N} x_{ti^*}^p = \sum_{t\in N\setminus\{j^*\}} x_{ti^*}^p \geq x_{i^*j^*}^p \tag{2.26}$$

Let $PT_x(i, k)$ denote the set of paths between s and i that carry flow of commodity k according to the flow variables x. Let $C(\alpha)$ be the cost per unit of flow of path α, namely, $C(\alpha) = \sum_{e\in\alpha} c_e$. Let $V_x(\alpha, k)$ be the fraction of commodity k that flows through path α according to solution x.

Using this notation, let $\bar{\alpha} = \text{argmax}\{C(\alpha) : \alpha \in PT_x(i^*, p) \cup PT_x(j^*, q)\}$. Without loss of generality, we will assume that $\bar{\alpha}$ belongs to $PT_x(j^*, q)$. Figure 2.3 shows a simplified graphical representation of this notation (not showing every node in each path). For now, assume that $\bar{\alpha}$ is unique. Details of the demonstration when there are multiple paths with the same cost will be discussed at the end of the proof.

Given that (y, x) is optimal, it is impossible to redirect flow of commodity q from $\bar{\alpha}$ to other path in $PT_x(i^*, p)$. That is to say, every path in $PT_x(i^*, p)$ is saturated in terms of sending

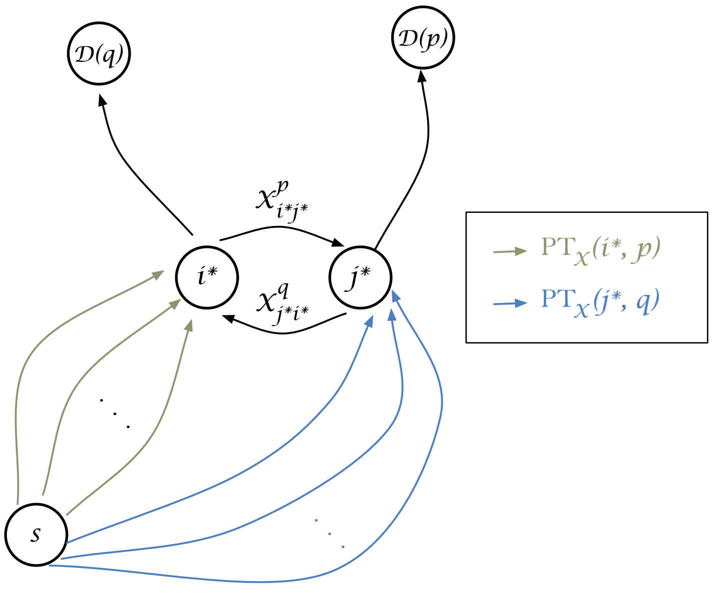


Figure 2.3: Simplified graphical representation of MCD_L

commodity q. In the linear relaxation, it holds that $y_{ij} = \max_k\{x^k_{ij}\}$, hence, increasing flow on a certain path increases the cost of the solution (in terms of fixed costs) only if such additional flow corresponds to the commodity with the highest fraction of flow. In this sense, the fraction of commodity q flowing through paths $PT_x(i^*, p)$ must be greater or equal than the fraction of commodity p, else there is still some slack to reduce flow of commodity q from the highest flow cost path $\bar{\alpha}$ and re-direct it to one of the paths in $PT_x(i^*, p)$. This implies that:

$$\sum_{\alpha \in PT_x(i^*,p)} V_x(\alpha, q) - \sum_{\alpha \in PT_x(i^*,p)} V_x(\alpha, p) \geq 0 \tag{2.27}$$

The first term of (2.27) is bound by all the flow of commodity q entering node i^*. That is to say:

$$\sum_{\alpha \in PT_x(i^*,p)} V_x(\alpha, q) \leq \sum_{t \in N} x^q_{ti^*} \tag{2.28}$$

Considering that $x^p_{j^*i^*} = 0$, there is no path in $PT_x(i^*, p)$ that contains arc (j^*, i^*). Therefore, we can even tighly bound the first term of (2.27) as follows:

$$\sum_{\alpha \in PT_x(i^*,p)} V_x(\alpha, q) \leq \sum_{t \in N \setminus \{j^*\}} x^q_{ti^*} \tag{2.29}$$

For the second term of (2.27), we have:

$$\sum_{\alpha \in PT_x(i^*,p)} V_x(\alpha,p) = \sum_{t\in N} x^p_{ti^*} = \sum_{t\in N\backslash\{j^*\}} x^p_{ti^*} \tag{2.30}$$

Therefore, we get:

$$\sum_{t\in N\backslash\{j^*\}} x^q_{ti^*} \geq \sum_{t\in N\backslash\{j^*\}} x^p_{ti^*} \tag{2.31}$$

Replacing (2.26) and (2.31) in (2.25) we finally obtain:

$$x^p_{i^*j^*} + x^q_{j^*i^*} \leq 1$$

This final result is valid for every edge, with either flow on one direction or both. Hence, we conclude:

$$\Rightarrow y_{ij} + y_{ji} \leq 1 \text{ for all } e = \{i,j\} \in E$$

Consider some technical details of the demonstration when there are multiple paths tied in the maximum costs in the set $\{C(\alpha) : \alpha \in PT_x(i^*,p) \cup PT_x(j^*,q)\}$. Suppose there is a path β in $PT_x(i^*,p)$ that has the same cost as $\bar{\alpha}$. If $c_{j^*i^*}$ is strictly positive, the argument that paths in $PT_x(i^*,p)$ are saturated in regard to commodity q is still valid. If $C_{j^*i^*}$ is equal to zero, then $F_{j^*i^*}$ must be positive. In such case, to redirect flow from $\bar{\alpha}$, there must exist another commodity $\bar{q}$ different from q such that:

$$y_{j^*i^*} = x^q_{j^*i^*} = x^{\bar{q}}_{j^*i^*}$$

In this way, when redirecting flow of commodity q from path $\alpha - (j,i)$ to path β, the variable $y_{j^*i^*}$ will not decrease its value and optimality will be maintained (if this is not true, then the initial hypothesis that (y,x) was optimal, is broken). In this scenario, a certain amount of flow can be redirected until eventually there are no more ties in the maximum costs path.

$\square$

Empirical and theoretical evidence show that among the proposed formulations, the $MCDR$ formulation delivers the best result in terms of the quality of its linear relaxation. However, as already discussed the size of its instances can be too large to handle them directly. Our proposal consist on finding an approximate solution to the $MCDR_L$ using a dual ascent method. Through this method, we can find a feasible dual solution that will serve as a lower bound to the optimal value and also reduce the problem size to be able to get a primal feasible solution. Section 3.1 describes the dual ascent framework and procedure, and section 3.2 focuses on how to use its output to get a primal feasible solution.

Chapter 3

Solution Approaches

3.1 Dual-Ascent Procedure

Since the SS-UND problem is NP-Hard and also reaches impeding sizes in real instances with the $MCDR$ formulation, we study the use of methods to find good lower bounds and heuristics solutions instead of the exact optimal solution. The structure of the network design problem makes it particularly attractive to apply dual-ascent routines that will iteratively improve a lower bound by finding better dual solutions. Furthermore, these dual solutions also serve to identify smaller feasible design networks that then can be used by other methods to find primal solutions. The dual-ascent procedure for the network design problem was first presented in [3], we base our implementation on such procedure adapting it specifically for the Directed Multicommodity formulation. We also provide empirical evidence on how the original method improves when it is designed to solve the directed problem instead of the undirected one.

Consider the linear programming dual for the formulation $MCDR_L$. Let v_i^k and w_{ij}^k be the dual variables associated respectively to contraints (2.20) and (2.18). For each commodity, one of the constraints of flow conservation (2.20) is linearly dependent to the rest, therefore we can fix $v_s^k = 0 \ \forall k \in K$. The dual formulation is the following.

Problem $DMCD$

$$\max\ z_D = \sum_{k \in K} v^k_{D(k)}$$

subject to

$$\begin{aligned} v^k_j - v^k_i &\le c^k_{ij} + w^k_{ij} \\ v^k_i - v^k_j &\le c^k_{ji} + w^k_{ji} \end{aligned} \qquad \forall \{i,j\} \in E, \forall k \in K \tag{3.1}$$

$$\begin{aligned} \sum_{k \in K} w^k_{ij} &\le F_{ij} \\ \sum_{k \in K} w^k_{ji} &\le F_{ji} \end{aligned} \qquad \forall \{i,j\} \in E \tag{3.2}$$

For any feasible solution of the $DMCD$, its objective function value constitutes a lower bound for the linear relaxation of the $MCDR$, and hence for the original integer problem. As explained in [3], the general strategy consists of iteratively modifying the values of the variables w^k_{ij}, w^k_{ji} (w-values), and v^k_i (v-values) in order to increase monotonically the value of the objective function, and thus improve the lower bound. Note that the objective function does not depend explicitly on the w-values. Considering this, the w-values could be fixed to a determined value that satisfies (3.2), and then the problem reduces to finding the best v-values that satisfies (3.1). Furthermore, with this strategy, v-values no longer depend on other commodities values. This allows us to decompose the problem in subproblems that can be solved individually by commodity. Let $SP_k(w)$ denote this subproblem corresponding to commodity k with the fixed w-values w. $SP_k(w)$ has the following structure:

Problem $SP_k(w)$

$$\max\ v^k_{D(k)}$$

subject to

$$\begin{aligned} v^k_j - v^k_i &\le c^k_{ij} + w^k_{ij} \\ v^k_i - v^k_j &\le c^k_{ji} + w^k_{ji} \end{aligned} \qquad \forall \{i,j\} \in E \tag{3.3}$$

The structure of the subproblem $SP_k(w)$ corresponds exactly to the structure of the dual of the classic Shortest Path Problem from origin s to destination $D(k)$ using as flow costs the modified costs $\hat{c}_{ij} = c^k_{ij} + w^k_{ij}$. Consequently, given the w-values, the length of the minimum path from the origin to each node i gives the v^k_i values that optimize the subproblem $SP_k(w)$. In particular, the optimal value of the objective function $v^k_{D(k)}$ is the length of the shortest path from the origin to the destination of commodity k. Thus, the value of the main dual problem $DMCD$ can be improved by increasing the w-values (and hence the $\hat{c}_{ij}$ values) so that they augment the length of a shortest path for a certain commodity.

Basically, the dual ascent method increases the objective function by iteratively increasing one or more appropriately chosen w-values in such a way that:

(i) Constraints (3.2) remains feasible.

(ii) The length of the shortest path s-$D(k)$, $v^k_{D(k)}$, increases at least for one commodity $k \in K$ in each iteration.

To satisfy the first condition, we consider as candidates to increase its value only those w^k_{ij} for which constraints (3.2) still have slack. Suppose that for a certain iteration, for a certain arc (i,j), $\sum_{k\in K} w^k_{ij} < F_{ij}$. We define $S_{ij} = F_{ij} - \sum_{k\in N} w^k_{ij}$ as the value of the slack in this particular constraint. This slack can be interpreted as the *unabsorbed fixed charge* that will iteratively be allocated in certain w-values to improve the lower bound Z_D.

The authors of this general algorithm (see [3]) developed two different algorithms that differ mainly in the selection of arcs where the slack is assigned, the *path diversion* method and the *labeling* method. We use the *labeling* method, which increases at each iteration several w-values of the same commodity k.

As discussed, the general idea is to increase as much as possible the lengths of the $s-D(k)$ paths for every commodity. The *labeling* method accomplishes this by partitioning the set of nodes N into two subsets for every commodity: $N_1(k)$, which always contains the source for commodity k, and $N_2(k)$, which always contains the destination $D(k)$ for commodity k. Nodes in $N_2(k)$ are also referred to as the *labeled* nodes for commodity k. If every arc in the *cutset* between $N_1(k)$ and $N_2(k)$ increases its w-values, then naturally the length of the shortest path between the source and every node in $N_2(k)$ will increase (including the destination $D(k)$). However, not every arc in such cutset must increase its w-value to obtain the same result. The *labeling* method uses this notion to identify the best possible arcs to increase its w-values and also increase multiple values simultaneously.

Let $A(k) = \{(i,j) \in A \ / \ i \in N_1(k) \wedge j \in N_2(k)\}$ denote the directed cutset between $N_1(k)$ and $N_2(k)$. The objective is to identify which of the arcs in $A(k)$ must increase its w^k_{ij} values to increase $v^k_{D(k)}$, the length of $s-D(k)$, the shortest path for commodity k. Note that if a certain arc (i,j) in $A(k)$ does not belong to $s-D(k)$, then an increase of w^k_{ij} will not increase $v^k_{D(k)}$. We refer to the arcs that do belong to the shortest path $s-D(k)$ as *tight arcs* (note that there could be multiple shortest paths with the same cost, in such cases, there could be more than one tight arc in $A(k)$). Tight arcs must satisfy the following condition:

$$v^k_j - v^k_i = c^k_{ij} + w^k_{ij} \tag{3.4}$$

If non-tight, we define the *tight slack*:

$$TS^k_{i,j} = (c^k_{ij} + w^k_{ij}) - (v^k_j - v^k_i) \tag{3.5}$$

We also use $A'(k) \subseteq A(k)$ to denote the set of every *tight arc* in $A(k)$, and $A''(k) = A(k) \backslash A'(k)$ to denote every *non-tight arc* in $A(k)$.

With this definitions, we can state that for a given $N_1(k)$ and $N_2(k)$, to increase $v^k_{D(k)}$ we must increase w^k_{ij} for every arc $(i,j) \in A'(k)$. Nevertheless, when doing so, besides increasing

$v^k_{D(k)}$, we also increase v^k_j for every $j \in N_2(k)$ which in effect, according to (3.5), will reduce the *tight slack* of every arc in $A''(k)$. Eventually, This could produce at a certain point, that a non-tight arc becomes tight, and thus, every additional amount increased in the w^k_{ij} values of the tight arcs will be of no use towards increasing the lenght of the shortest path to the destination $D(k)$. Essentially, an increase in the length of a certain arc that belongs to the shortest path will augment the final length of the shortest path only when such increase does not make another path a better alternative.

As already discussed, the increase of the w-values is also restricted by the feasibility constraints of $DMCD$. Hence, in accordance with the stated conditions (i) and (ii), to determine the maximum increase in the lengths of the arcs in $A'(k)$ the following must be considered:

(i) To maintain feasibility of (3.2), the maximum increase of the w-values in $A'(k)$ is given by the minimum slack available,

$$\rho_1 = \min \{S_{ij} : (i,j) \in A'(k)\} \tag{3.6}$$

(ii) To avoid tight arcs to stop being tight, the maximum increase of the w-values in $A'(k)$ is given by the minimum amount needed to make at least one non-tight arc become tight.

$$\rho_2 = \min \left\{TS^k_{i,j} : (i,j) \in A''(k)\right\} \tag{3.7}$$

An increase of $\rho = \min\{\rho_1, \rho_2\}$ in $w^k_{ij}, \forall (i,j) \in A'(k)$ produces that v^k_j augments in $\rho, \forall j \in N_2(k)$, maintaining feasibility in $DMCD$. At the end of each increase, if $\rho = \rho_1$, the slack S_{ij} of some arc $(i,j) \in A'(k)$ is reduced to zero. In the opposite case, if $\rho = \rho_2$, some non-tight arc of $A''(k)$, will move to $A'(k)$.

The dual-ascent procedure mechanizes this strategy. Initially, every w-value is fixed to zero and the v-values are calculated by solving a Shortest Path Problem from s to every destination $D(k)$ using the flow costs c_{ij} as arc lengths. Afterwards, considering the value v_i as the result obtained, each v^k_i is calculated by multiplying by the corresponding demand: $v^k_i = d_k \cdot v_i$. This first step can only be done by the fact that there is a single source structure; otherwise, multiple shortest paths routines should have to be performed. After setting the initial distances, the labeled node sets are initialized only including the destination for each commodity, $N_2(k) = \{D(k)\}$, that implies that $N_1(k) = N \backslash N_2(k)$, however, there is no need to store the non-labeled nodes. In each iteration the algorithm chooses a commodity k such that $s \notin N_2(k)$, then calculates de maximum possible increase and simultaneously reassigns the w-values of the arcs in $A'(k)$ along with the distances of the nodes in $N_2(k)$ (and hence, z_D, the dual objective function, is also updated). If such increase produces that for a certain arc $S_{ij} = 0$, then node i must be labeled. The algorithm iterates until s is labeled in each $N_2(k)$, $\forall k \in K$. This labeling procedure guarantees that, at the end, every arc with zero slack will constitute a feasible design solution which allows a directed path from s to every destination $D(k)$.

Finally, the dual-ascent procedure will deliver as an output, both the value of a feasible dual solution z_D, that constitutes a lower bound to the $MCDR$ problem, and also a subset of directed arcs that allows a feasible primal solution.

In the next section we describe our specific implementation based on this labeling procedure.

3.1.1 Implementation Details

In the previous section we already presented that by fixing the w-values, the dual problem $DMCD$ can be decomposed in subproblems that no longer have an explicit relation between commodities. This concept is a key element to the particular implementation that we developed.

The labeling method that we discussed maintains sets of labeled nodes $N_2(k)$ and sets of tight arcs $A'(k)$ and non-tight arcs $A''(k)$ in the cutset between the labeled and non-labeled nodes. There is one of each of these sets for every commodity k that must be stored in a determined data structure. In each iteration, a certain commodity must be chosen and the corresponding sets must be used. However, when the commodity is already chosen for a particular iteration, the information stored in the sets for the other commodities are of no use for that iteration. Considering this, there is no need to store these sets for every commodity simultaneously if the commodity choice is maintained until the destiny for such commodity is labeled. At this point, another commodity can be chosen, and the data structures to store the sets can be reset. Basically, the algorithm that we developed will choose only a certain commodity for each iteration. For such commodity, it will allocate as much of the slack as needed in the w-values of that commodity to increase the shortest path between the source and the sink. Only then it will iterate to the next commodity. This allows us to maintain in memory only one of each set instead of one per commodity, and in consequence save memory space. The cut set $A(k)$ and its corresponding subsets of tight and non-tight arcs can also be recalculated in each iteration to even save up more memory space. However, we opted to store them explicitly considering we only need to store them for one commodity.

Another important detail of our implementation is that it is based on the dual of the *Directed* Multicommodity formulation. In the original algorithm proposed in [3], there is only one slack S_{ij} for each edge $\{i, j\}$ that is used up by the w-values w_{ij}^k and w_{ji}^k. As imposed by constraints (3.2) of the DMCD formulation, the directed version that we propose has a slack for every arc. Therefore, for each edge $\{i, j\}$, there will be a slack S_{ij} for arc (i, j) that will be used up by the w-values w_{ij}^k and also another slack S_{ji} for arc (j, i) that will be used up by the w-values w_{ji}^k.

The network is modeled by a directed graph composed of a set of nodes holding an adjacent directed arc list. Each arc is a structure that contains its corresponding fixed cost F_{ij}, its remaining slack S_{ij}; and its flow cost per unit c_{ij}. Even though the MC formulation accepts different flow cost for each commodity, for this particular implementation we only need to store $c_{ij}, \forall(i, j) \in A$ and $d_k, \forall k \in K$ to obtain the flow costs $c_{ij}^k = d_k \cdot c_{ij}$.

The network data structure also stores the source node s, the demand for each commodity d_k and a reference to their destination nodes $D(k)$. The labeled nodes $N_2(k)$ are stored using a Boolean array and also a singly linked list to be able to access only the labeled nodes more quickly. The cut sets $A'(k)$ and $A''(k)$ are stored using circular doubly linked lists to be able to add or delete arcs from the lists more efficiently.

Considering the way that the algorithm labels the nodes, there is no need to store or update the w-values directly. As discussed in [3], after every ascent step, w-values satisfy the condition:

$$w_{ij}^k = \max\left\{0, v_j^k - v_i^k - c_{ij}^k\right\} \forall (i,j) \in A \tag{3.8}$$

Therefore, maintaining the v-values updated is enough and hence, the algorithm does not require to explicitly store the w-values. Moreover, given that each arc stores its slack S_{ij}, during the dual-ascent procedure, the w-values are only used to calculate the *tight slack* TS_{ij}^k (as per equation (3.5)). However, the first time an arc (i,j) appears in a k iteration, initially $w_{ij}^k = 0$, therefore the *tight slack* can be calculated only from the flow costs c_{ij}^k and the distances v_i^k and v_j^k. Assuming that the arc is non-tight ($TS_{ij}^k \neq 0$), after any w-value increase phase, the *tight slack* reduces exactly in the amount increased in that given iteration. Hence, instead of storing or calculating the w-values to be able to calculate the *tight slack*, it can rather be stored directly as an attribute of each element in the list of non-tight arcs $A''(k)$. This value will be local to $A''(k)$, making it unnecessary to store it for other arcs, and therefore not present as an attribute of the arc structure.

Considering the mentioned above, *Algorithm* 1 shows a pseudo code for this specfic implementation of the dual-ascent routine.

The dual ascent routine delivers as output: (a) a file containing all the zero slack arcs ($S_{ij} = 0$), which form a feasible solution for $MCDR$; and (b) a summary file, storing the value of Z_D, CPU time and number of iterations.

3.1.2 Algorithm Enhancements

The Dual-Ascent procedure iteratively chooses a certain commodity, explores the cut set $A(k)$, increases w-values in its maximum possible value and then updates the dual objective function z_D. Notice however, that the commodity choice is arbitrary (and moreover, in our specific implementation it is maintained until the destination node is labeled). Therefore, there is no guarantee that the slack S_{ij} is optimally allocated within the corresponding w-values w_{ij}^k for each arc. In [3] a strategy using the complementary slackness conditions is proposed to test the correct allocation of w-values.

The output of the dual ascent procedure provides a subset of arcs with zero slack that form a feasible design solution for the problem. That is $y_{ij} = 1$ if $S_{ij} = 0$. For this design, the optimal flow variables x_{ij}^k can be easily obtained by solving a shortest path problem for each commodity (using the corresponding costs c_{ij}^k as arc lengths). This feasible primal solution is very useful to identify certain conflictive arcs that might have their w-values using up the slack in a non-optimal allocation. The first step is to check the following complementary slackness conditions considering the dual w variables and the primal feasible solution (y, x) that can be generated from the dual ascent output.

$$\begin{aligned} w_{ij}^k(x_{ij}^k - y_{ij}) = 0 \\ w_{ji}^k(x_{ji}^k - y_{ji}) = 0 \end{aligned} \qquad \forall \{i,j\} \in A, \forall k \in K \tag{3.9}$$

```
***Step 0:*** General Initialization
Set $S_{ij} \leftarrow F_{ij}, \forall (i,j) \in A$.
Run a shortest path routine from $s$ to every node in the network; let $v_i$ be the shortest
path length to node $i$.
Set $v_i^k \leftarrow d_k \cdot v_i$.
Set $z_D \leftarrow \sum_{k \in K} v_{D(k)}^k$.
Initialize the set $CANDIDATES = K$, the set of *unmarked commoditites*.
while $CANDIDATES \neq \phi$ do
    Select a commodity $k$ from $CANDIDATES$.
    ***Step 1:*** Commodity Initialization
    Set $N_2 = \{D(k)\}$.
    Set $A' = \phi$ and $A'' = \phi$.
    forall the arcs $a = (i, D(k)) \in A$ do
        Calculate $TS_{ij}^k = c_{ij}^k - v_j^k + v_i^k$, $(j = D(k))$
        if $a$ is tight then
            add $a$ to $A'$
        else
            add $a$ to $A''$ with its corresponding TightSlack
        end
    end
    ***Step 2:*** Iterations for Commodity $k$
    while $s \notin N_2$ do
        Set $\rho_1 = \min\{S_{ij} : (i,j) \in A'\}$
        Set $\rho_2 = \min\{TS_{ij}^k : (i,j) \in A''\}$
        Set $\rho = \min\{\rho_1, \rho_2\}$.
        Update $S_{ij} \leftarrow S_{ij} - \rho, \ \forall (i,j) \in A'$.
        Update $v_i^k \leftarrow v_i^k + \rho, \ \forall i \in N_2$.
        Update $TS_{ij}^k \leftarrow TS_{ij}^k - \rho, \ \forall (i,j) \in A''$
        Increase $z_D = z_D + \rho$.
        if $\rho = \rho_1$ for some $(i^*, j^*)$ satisfying $S_{i^*j^*} = 0$ then
            Label node $i^*$ : $N_2 \leftarrow N_2 \cup \{i^*\}$
            Update arcs in cutsets $A'$ and $A''$, removing all arcs $(i^*, j)$ with $j \in N_2$, and
            adding all arcs $(l, i^*)$ with $l \in N \backslash N_2$ to the corresponding set
        end
        if $\rho = \rho_2$ for some $(i^*, j^*)$ satisfying $TS_{i^*j^*}^k = 0$ then
            Remove $(i^*, j^*)$ from $A''$ and add it to $A'$
        end
    end
    Remove $k$ from $CANDIDATES$
end
```

Algorithm 1: Dual-Ascent Implementation Pseudo Code

Naturally, in (y, x) there can not be flow if the corresponding design variable is zero, hence, the only way to break this conditions is when $y_{ij} = 1$ and $x_{ij}^k = 0$. In such case, in an optimal solution, w_{ij}^k must be zero. Therefore, inspecting the arcs with non-zero w-values, can help to identify certain conflictive arcs that are not meeting the complementary slackness conditions. The strategy to possibly improve the feasible primal solution consists on forcing the identified conflictive w-values to be zero. This is accomplished by re-executing the dual ascent procedure but assuming that S_{ij} is zero each time it comes across an specific arc with (i, j, k) indices that did not meet the conditions. The new dual solution produces a new primal feasible solution and the complementary slackness conditions can again be checked. This leads to an iterative scheme where in each step a new dual solution could be found. This strategy, however, does not guarantee that a better solution is actually found.

3.2 Primal Solutions

In terms of actually solving the original SS-UND problem, the dual-ascent procedure only provides a lower bound to its optimal objective function value and a subset of arcs that allow the existence of a feasible primal solution. In this section, we propose and discuss alternatives to get a primal solution from the given subset of arcs.

In [3], drop/add heuristics are used to find good primal solutions. We developed two alternative methods that consist of performing an intermediate processing of the network given by the dual-ascent and then solving either a MCD or a $DICUT$ formulation over such network.

3.2.1 MCD Formulation

The reduced network structure provided by the dual-ascent, which we denote $G' = (V, A')$ besides having significantly less arcs, can further reduce its size in terms of the MCD formulation.

Essentially, we take advantage of the fact that the new network has directed arcs instead of non-directed edges, which eventually produces that a given node becomes inaccessible by the origin. Identifying such nodes enables us to remove certain design and flow variables from the problem.

Let $i \to j$ denote that i is connected to j if and only if there exists a directed path from i to j. We also define A_i^+ and A_i^- respectively as the forward and backwards star of node i. That is, $A_i^+ = \{j \in N : (i, j) \in A\}$ and $A_i^- = \{j \in N : (j, i) \in A\}$

Given the way the algorithm labels the nodes, every labeled node is connected to at least a certain destination $D(k)$. However, there is no guarantee that they are accessible from s. If s is not connected to a certain node i, then we can remove from A every arc (i, j) with $j \in A_i^+$ and every arc (j, i) with $j \in A_i^-$ without loosing optimality, and hence reducing the problem size.

From another point of view, given the lower density of the sub-network delivered by the dual-ascent procedure, it also may occur that certain commodities cannot flow through every arc. Let $CON(k) = \{i \in N; i \to D(k)\}$ denote the set of nodes connected to the destination of commodity k. If $i \notin CON(k)$, then we can state that in every feasible solution of MCD

the variables x_{ij}^k are equal to zero for every arc (i,j) with $j \in A_i^+$. Therefore, we can remove such flow variables and also the constraints (2.18) corresponding to those arcs and commodity from the formulation. These reductions significantly decrease the size of the problem and in some cases it is possible to solve it directly with traditional linear programming software through the MCD formulation.

3.2.2 DICUT Formulation and Rounding Heuristic

Considering that G' has a lower density than G, the $DICUT$ formulation becomes an interesting alternative to get a primal solution. Given that G' has less amount of arcs, the number of cutting planes needed to find a feasible solution will be decreased along with the iterations of the method.

Additionally, as in the last subsection, we use a similar approach to strengthen the formulation. Let $T_i = \{t \in T / i \to t\}$ denote the set of all destination nodes that are reachable by a certain node i. In any feasible solution, the maximum volume that may flow out of a node i is given by the total amount of demand of the destination nodes reachable by i. Therefore, we can state that $\forall i \in N$ each arc (i,j) with $j \in A_i^+$ will have a flow bound by $M_{ij} = \sum_{t \in T_i} d_t$. In consequence, we can replace constraints (2.13) with:

$$x_{ij} \leq M_{ij} \cdot y_{ij} \qquad \forall (i,j) \in A \tag{3.10}$$

Unlike the STP, where the $DICUT_L$ and MC_L formulations are equivalent, in the SS-UND, even with the proposed strengthening, the $DICUT_L$ can present a higher gap than the MC_L due to fractional values on the design variables.

To address this problem, we propose a *Rounding Heuristic* to execute after the $DICUT$ has ended adding all the possible cuts to the problem to be able to get a feasible integer primal solution. This heuristic is a generalization of the *Minimum Cost Paths Heuristic* developed for the STP in [16] and discussed in [18]. In essence, the heuristic which we denote MPH, tries to incorporate information both from the LP fractional solution and the cost structure of the network.

The complete cutting plane procedure and heuristic is the following:

Cutting Plane Method

- *Step 0*: Initialization:

    1. Load graph from dual ascent solution.
    2. Eliminate unfeasible arcs.
    3. Calculate T_i, for all $i \in N \setminus T$.
    4. Calculate M_{ij}.

5. Formulate DICUT LP.

- *Step 1*: Solve LP.
- *Step 2*: For each $i \in T$:
  1. Find max-flow from s to $i/$.
  2. If max-flow > 1, add violated cut.
- *Step 3*: If no cuts were added, continue to *Step 4*. Else, go to *Step 1*.
- *Step 4*: If LP solution is not integer, run rounding heuristic.

Rounding Heuristic

- *Step 0*: Initialization:
  1. Eliminate every arc with $y_{ij} = 0$ from A
  2. Set $\bar{r} \leftarrow \frac{1}{|K|}(\sum_{k\in K} d_k)$.
  3. Set $\hat{c}_{ij} \leftarrow (1 - y_{ij}/2)(c_{ij}\bar{r} + F_{ij})$.
  4. Set $UMK = K$. Set $M = \{s\}$.
  5. Calculate shortest path tree from s to every node in N using costs $\hat{c}_{ij}$. Let v_i be the path length to node i.
- *Step 1*: Find $k^* = \arg\min\{v_{D(k)}/k \in UMK\}$.
- *Step 2*: Set $UMK = UMK \setminus \{k^*\}$. Set $\bar{r} = \frac{\bar{r}(|UMK|+1)-d_{k^*}}{|UMK|}$.
- *Step 3*: For every arc (i,j) in the path $s - k^*$, update $\hat{c}_{ij} \leftarrow \bar{r} \cdot c_{ij}$ and mark it. Add to M each node i in this path , including $D(k^*)$, and update v_i according to the new $\hat{c}_{ij}$ values.
- *Step 4*: Recalculate shortest paths to every node in $N \setminus M$.
- *Step 5*: If $UMK \neq \phi$, go to Step 1. Else STOP.

The final feasible primal solution will be the tree formed by every marked arc.

Chapter 4

Computational Results

Every test described in this section was performed in a personal computer with an Intel Core 2 Duo 2.66Hz processor with 4GB RAM. To solve the linear programming models with traditional methods we used *Gurobi Optimization* mathematical programing solver software version 5.5 [7].

To test the methods and algorithms we used randomly generated networks with a grid structure of different sizes. Considering the original nature of the problem, the networks have non-directed edges $\{i, j\}$ that have equal flow and fixed costs in both directions ($c_{ij} = c_{ji}$ and $F_{ij} = F_{ji}$). Both of the costs are randomly assigned with a uniform probability distribution. The relation between the maximum possible fixed cost (F_{max}) and the maximum possible flow cost (C_{max}) is modified to study the effect on the algorithm performance. Two sets of random instances are generated by modifying this relation; set A of random networks have a F_{max}/C_{max} of 1.67 and set B a relation of 3.33. Given the number of nodes, a squared grid is constructed with each side having L nodes, where $L = \left\lfloor \sqrt{N} \right\rfloor$. If the square root of the number of nodes is not an integer number, then an additional row of $(N - L^2)$ nodes is added to the grid. Every arc always exist, connecting each node to other adyacent nodes in the same row and upper or lower rows. That is to say, that the number of edges is constant between instances with the same number of nodes. The source node s and the set of destination nodes T are randomly assigned between all available nodes.

In every test, we vary the size of the instances to analyze the scaling of the methods. We also include *manageable* size instances to be able to run a MIP model in order to obtain the integer optimal of the problem, and thus, be able to identify the lower and upper bound gap.

Table 4.1 shows the size of the problems that we used and their equivalence on the MCD formulation. We used instances ranging from 100 nodes, 360 arcs and 20 commodities to others of 2000 nodes 7820 arcs and 1600 commodities.

			Continuous	Integer	Total
N	K	A	Variables	Variables	Constraints
100	20	360	7,200	360	9,200
100	80	360	28,800	360	36,800
500	200	1910	382,000	1,910	482,000
500	400	1910	764,000	1,910	964,000
1000	200	3872	774,400	3,872	974,400
1000	800	3872	3,097,600	3,872	3,897,600
2000	1600	7820	12,512,000	7,820	15,712,000

Table 4.1: Test problems and equivalent MCD size

4.1 Directed Dual-Ascent

As mentioned on section 3.1.1, we implemented a *directed* version of the dual-ascent procedure according to the strengthening discussed in section 2.2. In this section, we show results of tests that compare the original non-directed dual-ascent with the directed version. To measure the performance of the different methods we calculated the exact optimal value for each instance. We used the MCD formulation for this purpose. We repeated each test 10 times for each selection of parameters (N, K and network of type A or B), generating 10 different instances. Table 4.2 shows the average gap for each method for the different instances. The same information is also shown graphically in Figure 4.1.

		A-Type		B-Type	
N	K	GAP ND [%]	GAP D [%]	GAP ND [%]	GAP D [%]
100	20	4.64	2.83	10.62	6.62
100	80	3.44	2.50	8.53	5.84
500	200	3.73	2.86	7.90	5.18
500	400	2.58	2.04	6.46	4.99
1000	200	4.56	3.55	8.87	6.54
1000	800	2.24	1.82	5.39	4.45

Table 4.2: Directed and Non-Directed Dual-Ascent GAP comparison

As expected and for every instance that we tested, the directed version of the dual-ascent clearly outperforms the non-directed version.

The results also show that the dual-ascent procedure performs better in networks with a lower Fixed/Flow cost relation (A-type).

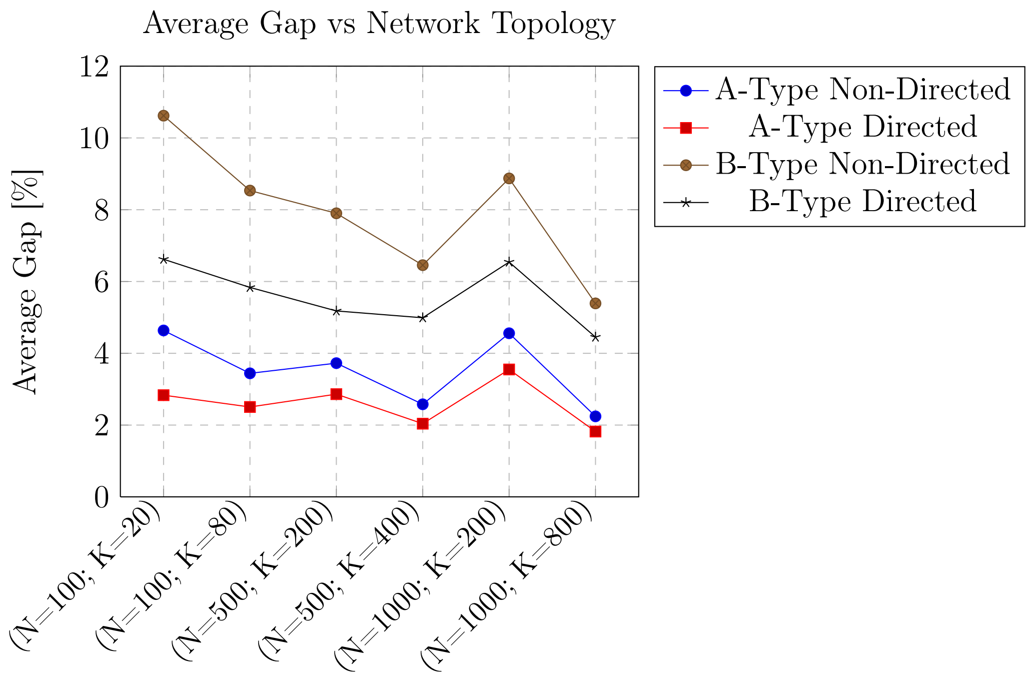


Figure 4.1: Performance Comparison of Directed and Non Directed DualAscent

4.2 Commodity Order

The algorithm described in 3.1.1 does not specify the order in which commodities are chosen. In previous studies [3] it was reported that specific rules to choose the order of commodities does not significantly or consistently improve the performance of the algorithm. However, our particular scheme of maintaining the commodity choice until its destination node is marked suggested that the order could play an important role in the performance. We tested different rules and observed a consistent improvement of the mean gap when choosing commodities sorted from highest demand volume to lowest demand volume. To perform these tests, we used instances of the same sizes and characteristics as those of the previous section. We again used the MCD formulation to get an objective comparable value for the dual-ascent output, and we also repeated the process 10 times for each selection of parameters. Table 4.3 shows the average gap for each method for the different instances. The same information is also shown graphically in Figure 4.2.

Although the sorted scheme does show a significant improvement considering the average results in all the tested problems, not every instance result actually presented improvement. In some particular cases, the initial random sorting of demand delivered a smaller gap. This suggests that, for every instance, a specific sorting scheme that delivers a better result can be found. A possible procedure could be to parallelize the dual-ascent algorithm with a set of different sorting methods and then choose the highest objective function value obtained. However, considering the characteristics of the CPU used in this study we preferred to maintain the proposed sorting method for the following tests.

		A-Type		B-Type	
N	K	GAP Non-Sort [%]	GAP Sort [%]	GAP Non-Sort [%]	GAP Sort [%]
100	20	2.99	2.43	6.34	5.90
100	80	2.52	1.93	5.08	4.74
500	200	2.85	2.39	5.31	4.94
500	400	2.19	1.68	4.96	4.13
1000	200	3.52	2.88	6.54	5.89
1000	800	2.03	1.69	4.45	3.07

Table 4.3: Sorted and Non-Sorted Dual-Ascent GAP comparison

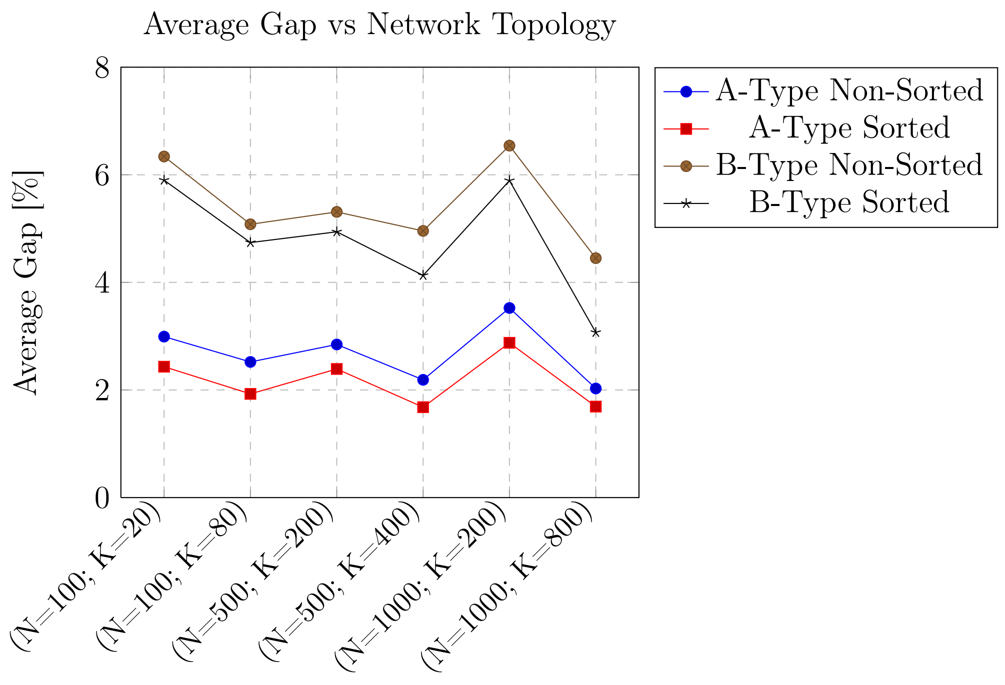


Figure 4.2: Performance Comparison of Sorted and Non Sorted DualAscent

4.3 Method Comparison

In this section we present results for every method that was previously discussed. For the dual-ascent procedure, we sorted the demands according to the improvement proposed in 4.2. We discarted the smaller sized sets of 100 nodes and also included a larger 2000 node, 1600 commodities, 3360 arcs set where the MCD MIP formulation could not be solved.

We again used multiple instances and show results for the average values for each category. Every instance was solved by each of the discussed methods according to the following scheme. Figure 4.3 shows a simplification of how the methods are applied according to this scheme. Tables 4.4 and 4.5 show the average results for the tested instances and Figure 4.4 shows the average gap of each method where the MIP solution could be obtained.

In first place, if the size of the instance allows it, we used the MCD MIP formulation to get the exact optimal value. Considering that in some cases the lower bound gap provided

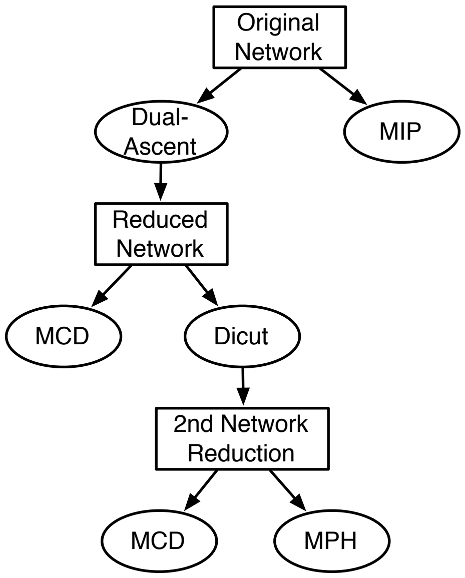


Figure 4.3: Method application scheme

by the dual-ascent was too large, this step becomes particularly relevant. With the exact optimal value we can measure the real performance of each method without depending on the quality of the dual-ascent result. However this was not possible in all cases, resulting in the reduction of this analysis only for smaller instances. We refer to this result as the MIP value and we report the CPU time to obtain it.

Secondly, we perform the dual-ascent procedure storing both the lower bound value and the subset of arcs that allow a feasible solution. We denote this lower bound as the DA value. For this test, we report the mean gap between the MIP value and the DA value obtained, along with the CPU time and the amount of arcs left in the sub-network as a percentage of the original amount of arcs. Using the sub-network provided by the dual-ascent procedure, we then get a feasible primal solution according to the process described in section 3.2.1 which consists of an arc elimination step and then solving a reduced size MCD formulaion to obtain an upper bound to the optimal value. We report both gaps of this value with respect to the DA value and the MIP value, along with the CPU time of the procedure.

Taking the sub-network that was used in the previous step, we then perform the DICUT method as described in section 3.2.2. This method provides an upper bound to the optimal value and we again report both its gap with respect to the DA value and with respect to the optimal value. The MPH heuristic performed in this step to find a integer feasible solution only takes into consideration the arcs that end up with a positive design value $y_{ij} > 0$. This subset of arcs can again be considered as a sub-network that allows a feasible solution that can be obtained with an alternative method other than the MPH rounding heuristic. For this method we also report the mean number of dicut iterations, the mean CPU time and the amount of arcs of the output sub-network as a percentage of the original amount of arcs.

Using the output sub-network of the dicut method we finally solve a MCD formulation. The value obtained with this method is the best possible value that the MPH heuristic can achieve and in a sense, measures the potential of the dicut formulation. As usual, for this method we report both of the gaps and the CPU time.

			MIP	Dual-Ascent			MCD (DA out)		
N	K	Type	time [s]	Gap [%]	time [s]	Arcs left [%]	Gap(da) [%]	Gap(mip) [%]	time [s]
500	200	A	69.8	2.31	0.8	34.4	2.48	0.11	14.5
		B	67.9	5.22	0.8	39.3	5.52	0.03	18.0
500	400	A	93.5	1.62	1.1	41.4	1.67	0.02	26.3
		B	122.8	4.05	1.1	46.0	4.15	0.03	35.7
1000	200	A	180.8	2.29	1.1	29.8	2.44	0.10	21.0
		B	188.5	5.19	0.5	38.8	5.50	0.11	30.7
1000	800	A	595.1	1.20	2.3	42.8	1.23	0.02	103.4
		B	899.3	4.36	3.4	46.3	4.43	0.09	128.1
2000	1600	A	-	-	8.3	43.9	1.15	-	761.0
		B	-	-	8.6	53.7	2.56	-	1195.4

Table 4.4: Final Result Summary: MIP, Dual-Ascent and Reduced MCD

			Dicut+MPH					Dicut+MCD		
N	K	Type	Gap(da) [%]	Gap(mip) [%]	iterations [#]	time [s]	arcs left [%]	Gap(da) [%]	Gap(mip) [%]	time* [s]
500	200	A	3.75	1.53	251.2	217.3	27.3	2.51	0.26	9.3
		B	11.03	6.40	329.0	422.1	27.8	5.58	0.66	9.4
500	400	A	2.61	1.03	122.6	135.3	34.1	1.68	0.09	17.4
		B	8.27	4.56	142.7	198.7	34.7	4.16	0.28	20.7
1000	200	A	4.07	1.87	1012.9	8004.6	22.2	2.50	0.27	12.4
		B	13.53	9.04	1919.3	43088.8	26.3	5.62	0.73	13.8
1000	800	A	2.63	1.46	268.2	2003.8	35.0	1.24	0.06	69.6
		B	11.09	7.22	343.0	2398.4	34.7	4.45	0.29	73.6
2000	1600	A	2.60	-	490.0	23160.8	35.5	1.16	-	422.0
		B	8.98	-	901.0	66713.5	38.6	2.58	-	407.4

(*) Execution time does not include the Dicut method.

Table 4.5: Final Result Summary: Dicut with MPH and MCD

In terms of the Dual-Ascent procedure, it can be observed that it takes significantly less CPU time to execute it in comparison to the other methods. In cases where the optimal solution could be obtained, the mean lower bound gap is always below 5.3% and tends to be lower when the fixed/flow cost ratio is smaller (type A networks) or when the relation between total nodes and demand nodes is smaller. The subnetwork this process delivers always has less than 54% of the arcs of the original network.

Solving the instances with the reduced size MCD formulation over the output network of the Dual-Ascent is, in a way, a tool to measure the quality of the delivered arcs in terms of their capacity to obtain a value as closer as possible to the MIP value. In this sense, the Dual-Ascent is an excellent arc selecting procedure. The mean upper bound gap obtained through the reduced MCD is always below 0.11% (when it could be measured). From another

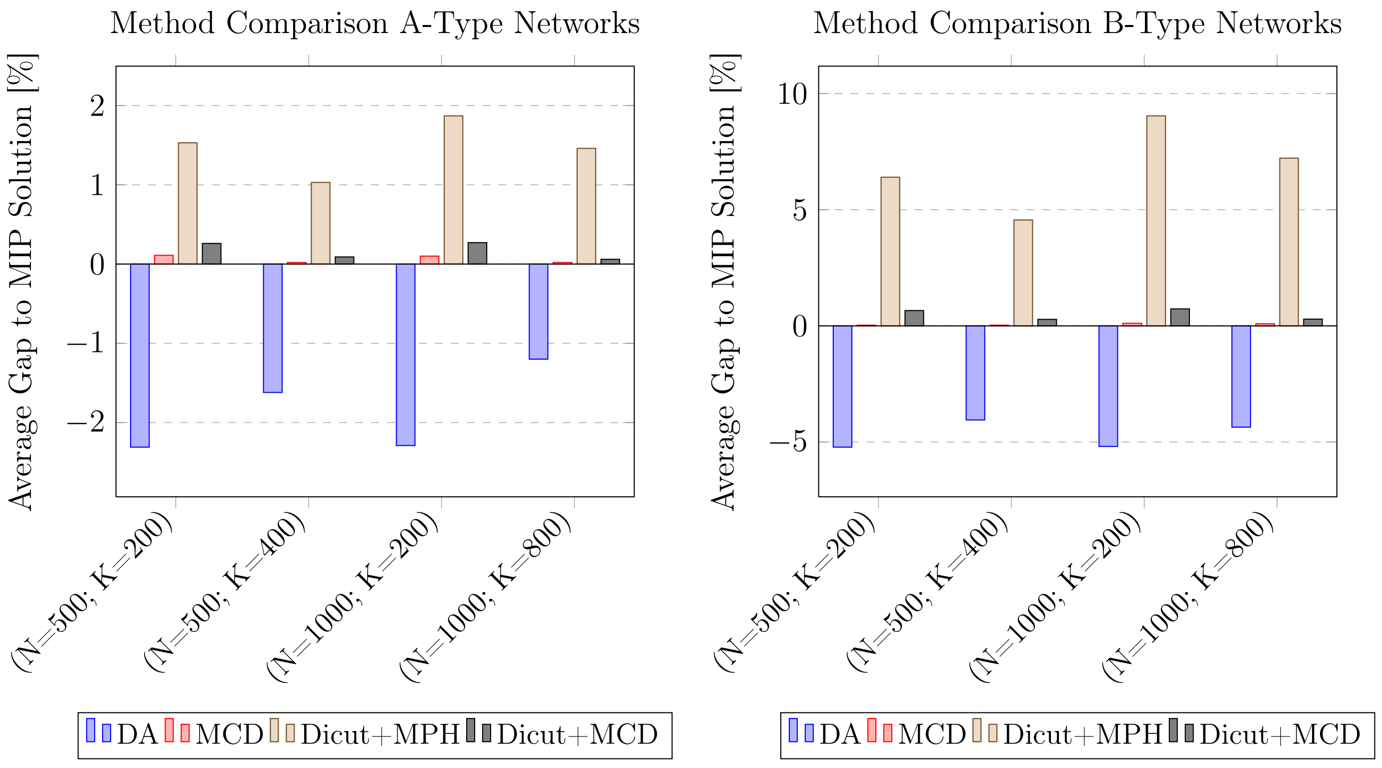


Figure 4.4: Average lower/upper bound gap of the applied methods

point of view, considering that in some cases the original problem size is unmanageable, the dual-ascent procedure allows to reduce the size of the problem enough to be able to solve it through the multi-commodity directed formulation. Throughout all the tested cases, a maximum gap of 5.5% is observed. However, in even larger instances this method might not be an option.

The Dicut method is significantly slower than the reduced size MCD and even slower than the original sized MCD MIP when it could be solved. This is a consequence of the amount of iterations needed for this method to converge. In every iteration $O(K)$ max flow/min cut problems are solved to identify the cuts to add to the formulation, and then a Dicut formulation LP is solved. The number of these iterations tends to be bigger in Type B networks and in test cases where the relation between total nodes and demand nodes is smaller. Even when the method could not identify any further cuts to add to the problem, the final LP solution was not integer in any case and the MPH rounding heuristic had to be used to find an integer solution. The CPU time taken by the heuristic is included in the total Dicut method time and is always significantly faster in relation to the Dicut iterations times (less than 1 second). In section 4.4 we study the impact on the final solution when the iterations are stopped before the method converges.

Although slower, the Dicut method solves a single commodity formulation LP that requires less memory space to store an instance than the multi-commodity formulation. This allows finding upper bounds to problems that are otherwise unapproachable with the previous methods. Through this approach, a mean gap of approximately 6.86% in relation to the Dual-Ascent lower bound is obtained. However, in relation to the actual MIP value, a gap of 4.14% is obtained. Again, this method has a worse performance when tested on networks with higher relation between fixed and flow costs (Type B). In such cases, the mean gap in relation to the MIP value is 6.8% when the mean gap is only 1.5% on the Type A networks. These results show such poor performance of the MPH rounding heuristic in Type B networks that it should not be considered as a viable option for finding good quality upper bounds to the SS-UND problem. Nevertheless, as already mentioned, the input network for the heuristic (arcs with $y_{ij} > 0$ after the dicut iterations) can instead be used to solve a MCD LP. Across all the tested instances, such network had an average of only 32% of the initial arcs, which is a network small enough to solve the LP in relatively short CPU times. Through this method, we obtain an average gap of 3.15% when compared to the Dual-Ascent lower bound and an average gap of 0.33% when compared to the actual optimal value. On one hand, it can be concluded out of these results that the MCD LP over the filtered network can deliver good upper bounds to the optimal value of the problem. On the other hand, and more importantly, we can conclude that both the Dual-Ascent and the Dicut are excellent arc selecting procedures.

4.4 Further Analysis - Dicut Performance

Considering the large CPU times required to perform the dicut method, we studied how the quality of the solution is improved with each iteration. As previously discussed, the final MCD method delivers the best possible primal solution that can be obtained with any heuristic from the dicut subset of arcs with positive design variables. By solving a MCD formulation after each dicut iteration, we are able to measure the real impact of each iteration independently of the quality of the chosen rounding heuristic.

To perform this test, we used two network topologies; one of 300 nodes, 565 arcs and 240 commodities and another of 500 nodes, 955 arcs and 400 commodities. For each case, we repeated the process 20 times generating a different random instance for each iteration. Figures 4.5 and 4.6 show curves for CPU time and intermediate gap for each iteration for each case.

It can clearly be observed that the method does not require performing all the iterations until no cuts are found. In both cases there is a point where no further improvement can be obtained even though the algorithm can still identify certain cuts that do not satisfy the flow condition. This suggests that a stopping rule could be implemented to reduce CPU time on larger sized instances. For example, periodically after a specified number of iterations, the rounding heuristic or an MCD formulation problem can be executed. If the gap improvement is lower than a given threshold, then the method can stop.

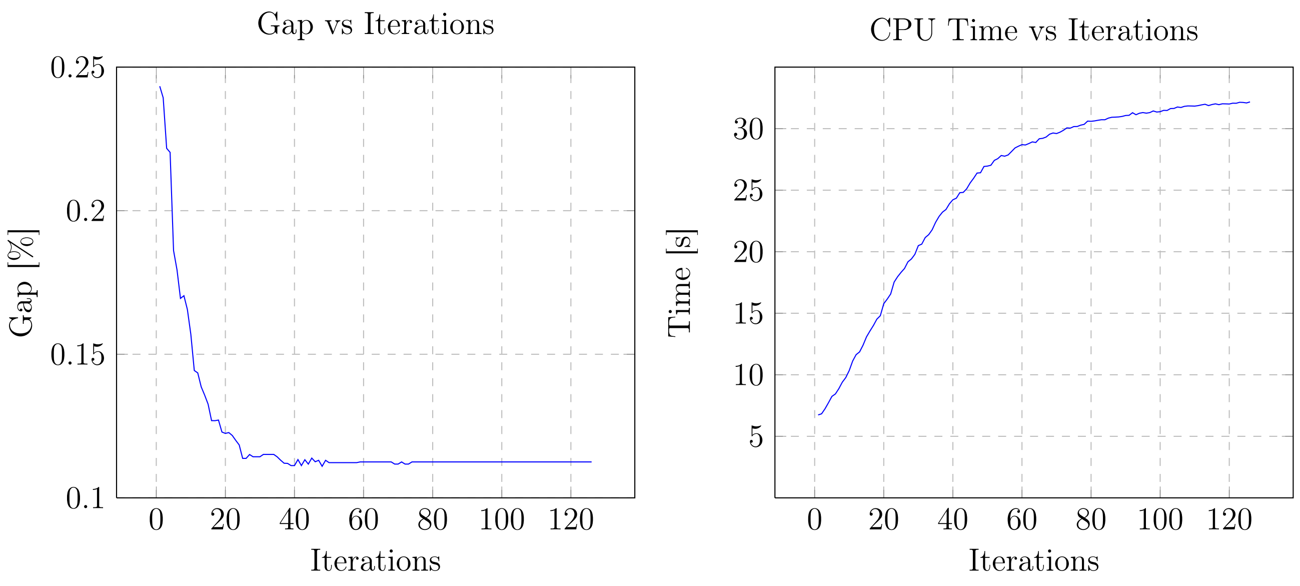


Figure 4.5: Dicut performance 300 nodes, 565 arcs and 240 commodities

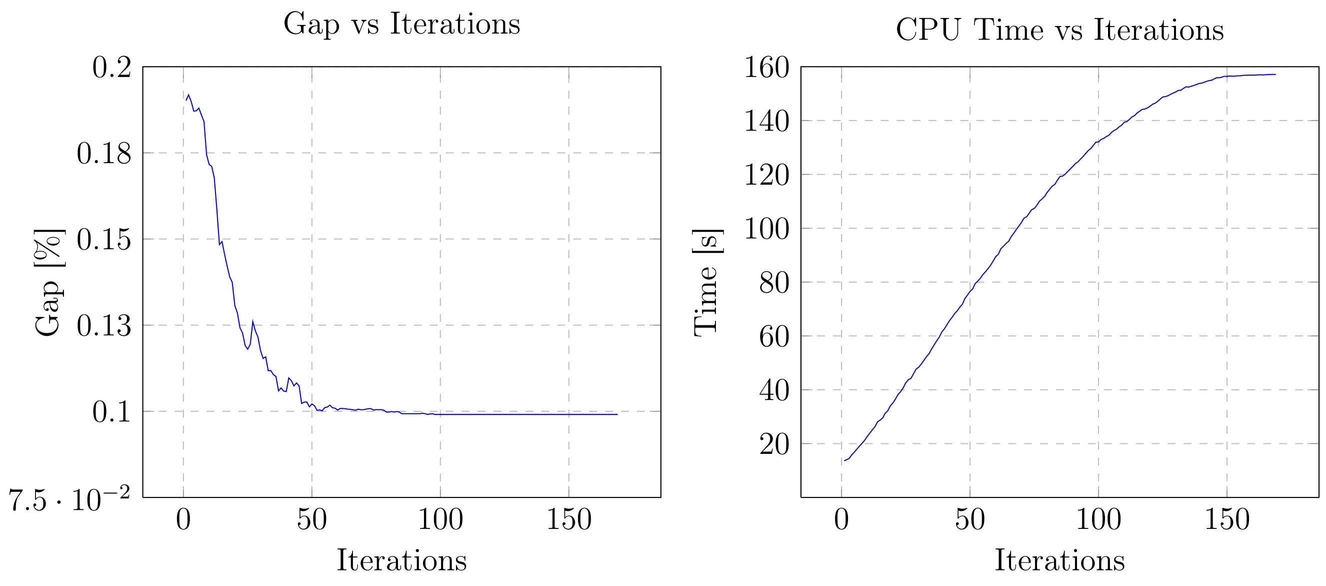


Figure 4.6: Dicut performance 500 nodes, 955 arcs and 400 commodities

Chapter 5

Conclusion

The results of this study show that single source uncapacitated network design problems can be solved effectively by combining extended multicommoity formulations, dual-ascent, a row generation scheme and rounding heuristics. Previous evidence shows that disaggregate multicommodity formulations reduce the gap between the discrete solution and the lineal relaxation solution. We show that by directing the design variables on a non-directed network an even tighter gap can be obtained. This result generalizes previous studies made on other problems like the Steiner Tree Problem.

To solve large-scale instances, we use a heuristic approach to find lower and upper bounds to the optimal of the integer problem. For the lower bound, we design a specialized dual-ascent procedure based on the work in [3]. We incorporate three main features to the algorithm; a directed design variable nature, a single source structure and a commodity sort scheme based on demand volume. We show empirical tests that corroborate a better performance of the procedure when incorporating these features. We also propose an alternative implementation that allows us to reduce the memory usage when storing variables of the algorithm without increasing computational effort.

In addition to the lower bound value, the dual-ascent procedure also delivers a reduced sized sub-network that can be used to find upper bounds to the problem. We use this sub-network to obtain good feasible primal solutions that serve as upper bounds to the problem. On the tested instances we observed that an average of 40% of the arcs were selected. We show that these arcs are of excellent quality in terms of their capacity to obtain a tight gap in relation to the optimal value. An average gap of 0.06% was obtained using the same formulation as in the original complete network.

To get a primal solution we used two different approaches; solve the multicommodity formulation over the selected arcs, and apply a row generation scheme based on a dicut formulation of the problem. The key difference between them is how the trade-off between execution time and CPU memory usage is managed.

On one hand, given the good quality of the sub-network obtained with the DA procedure, the first approach delivered excellent results. Across all the tested instances of different sizes and parameters, we obtained mean gaps between 1.0% and 5.5% (comparing primal and dual-ascent values). The best results where obtained with a smaller relation between fixed and flow costs and with a smaller relation between total nodes and demand nodes. However, this approach might still not be a feasible alternative in terms of its problem size in large instances.

On the other hand, the row generation scheme based on the dicut formulation offers a much smaller representation of the problem, which makes it a better alternative for large-scale instances. The procedure iteratively solves a linear relaxation of a dicut formulation, which requires significantly less memory space considering its aggregate demand nature. Given that the final solution obtained through the row generation method was never integer in its design variables, we developed a rounding heuristic. With this heuristic we obtained a mean gap of 6.9% across the tested instances. However, the performance of this method is considerably affected by the ratio between flow and fixed costs. On larger ratio instances, a mean gap of 10.6% was obtained while on smaller ratio instances a gap of 3.1% was obtained. These results suggest that a better heuristic should be developed for each type of instance. Continuing this string of thought, we studied the quality of the output of the dicut method in terms of its capacity to get a good upper bound independently of the heuristic. Essentially, the heuristic only uses a subset of arcs according to the values of the design values (arcs such that $y_{ij} > 0$). We used this subnetwork to solve a multicommodity directed formulation and thus we where able to obtain the best result any heuristic could obtain.

The subnetwork has approximately 30% of the arcs of the original network and across the tested instances we obtained mean gaps between 1.0% and 5.5% in relation to the dual-ascent value and mean gaps between 0.1% and 0.7% in relation to the actual optimal value. With the presented results we can conclude that the row generation scheme is also an excellent arc-selecting tool and there is a great potential to develop topology specific heuristics to find integer primal solutions in a faster way.

While this approach makes handling large-scale instances possible, it also takes considerably more CPU time given the amount of iterations it has to perform to converge. Measured across all the tested instances, the row generation approach took an average of approximately 200 times more CPU time to execute in relation to Multicommodity formulation approach. Considering this, we also studied how the number of iterations of the method affected the quality of the solution. As a result of this exercise we highlight that the method presents a fast convergence curve that suggests the use of a stopping rule. We believe this is a key finding that points towards the enhancement of the process that is taking the longest CPU time within method. Much remains to be studied in the separation problem; hence as topic of future research, we propose to study in greater detail the row generation scheme, either focused on better ways of finding violated cuts, other type of simple cuts or stopping rules to optimize the trade-off between CPU time and quality of the solution.

Chapter 6

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] Y.P. Aneja. Solving the steiner tree on a graph using branch and cut. *Networks*, 10:167–168, 1980.

[3] Anantaram Balakrishnan, Thomas L. Magnanti, and Richard T. Wong. A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design. *Operations Research*, 37(5):716–740, 1989.

[4] Francisco Barahona and Laszlo Ladányi. Branch and cut based on the volume algorithm: Steiner trees in graphs and max-cut. *RAIRO - Operations Research*, 40(1):53–73, 2006.

[5] Sunil Chopra, Edgar R. Gorres, and M.R. Rao. Solving the steiner tree problem on a graph using branch and cut. *ORSA Journal on Computing*, 4(3):320–335, 1992.

[6] Sunil Chopra and M. R. Rao. The steiner tree problem i: Formulations, compositions and extension of facets. *Math. Program.*, 64(2):209–229, April 1994.

[7] Inc. Gurobi Optimization. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2014.

[8] Kaj Holmberg and Johan Hellstrand. Solving the Uncapacitated Network Design Problem by a Lagrangean Heuristic and Branch-And-Bound. *Oper. Res.*, 46(2):247–259, 1998.

[9] T. Koch and A. Martin. Solving steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998.

[10] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18:1–56, 1984.

[11] Thomas L. Magnanti and Laurence L. Wolsey. *Handbooks in OR & MS*, volume 7,

chapter 9 Optimal Trees, pages 503–615. Elsevier Science B.V., 1995.

[12] Marcelo Olivares. Ascenso Dual en la Resolución de Redes de Gran Tamaño Aplicado a Diseño de Caminos Forestales, 2000. Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Departamento de Ingeniería Industrial, Santiago de Chile.

[13] Francisco Ortega and Laurence Wolsey. A branch-and-cut algorithm for the single commodity uncapacitated fixed charge network flow problem. *Networks*, 41(3):143–158, 2000.

[14] R. Rardin and U. Choe. Tighter relaxations of fixed-charge network flow problems. *Report J-79-18*, 1979.

[15] Ronald L. Rardin and Laurence A. Wolsey. Valid inequalities and projecting the multicommodity extended formulation for uncapacitated fixed charge network flow problems. *European Journal of Operational Research*, 71(1):95–109, November 1993.

[16] H. Takahashi and A. Matsuyama. An Approximate Solution for the Steiner Problem in Graphs. *Math.Japonica*, 24:573–577, 1980.

[17] Tony J Van Roy and Laurence A Wolsey. Valid inequalities and separation for uncapacitated fixed charge networks. *Oper. Res. Lett.*, 4(3):105–112, October 1985.

[18] Pawel Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.