Branching Rules for Minimum Congestion Multi-Commodity Flow Problems

Cameron Megaw
Clemson University, cmegaw@clemson.edu

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses
Part of the Applied Mathematics Commons

Recommended Citation
https://tigerprints.clemson.edu/all_theses/1488

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
Branching Rules for Minimum Congestion Multi-Commodity Flow Problems

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science
Mathematics

by
Cameron R. Megaw
August 2012

Accepted by:
Dr. Pietro Belotti, Committee Chair
Dr. Douglas Shier
Dr. Matthew Saltzman
Abstract

In this paper, we examine various branch and bound algorithms for a minimum congestion origin-destination integer multi-commodity flow problem. The problem consists of finding a routing such that the congestion of the most congested arc is minimum. For our implementation, we assume that all demands are known a priori.

We provide a mixed integer linear programming formulation of our problem and propose various new branching rules to solve the model. For each rule, we provide theoretical and experimental proof of their effectiveness.

In order to solve large instances, that more accurately portray real-world applications, we outline a path formulation model of our problem. We provide two methods for implementing our branching rules using branch and price.
Contents

Abstract ii

List of Figures v

Chapter 1. Introduction 1

Chapter 2. Basic Concepts 3
  2.1. Concepts of Graph Theory 3
  2.2. Network Flow Problems 4
  2.3. Computational Complexity 9
  2.4. Branch and Bound 11

Chapter 3. Minimum Congestion Model 14
  3.1. Model 14
  3.2. Complexity of the Problem 16
  3.3. Complexity for a Single Commodity 18
  3.4. Difficulties with the Model 19

Chapter 4. Contributions 21

Chapter 5. Developing Branching Rules 23
  5.1. Constructing a Branching Rule 23
  5.2. Considerations for Developing a Branching Rule 29

Chapter 6. Branching Techniques 32
  6.1. Terminology 32
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.1</td>
<td>Small directed graph</td>
<td>3</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Illustrations of graph definitions</td>
<td>4</td>
</tr>
<tr>
<td>2.1.3</td>
<td>A directed-out-tree</td>
<td>4</td>
</tr>
<tr>
<td>2.2.1</td>
<td>A small example of a multi-commodity flow problem</td>
<td>7</td>
</tr>
<tr>
<td>2.4.1</td>
<td>A small example of an ODIMCFP</td>
<td>13</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Graphing a multi-processor problem</td>
<td>17</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Example of a feasible path with infeasible cyclic flow</td>
<td>20</td>
</tr>
<tr>
<td>5.1.1</td>
<td>A partition of $st$-cut arcs</td>
<td>25</td>
</tr>
<tr>
<td>5.2.1</td>
<td>A balanced branch and bound tree</td>
<td>30</td>
</tr>
<tr>
<td>5.2.2</td>
<td>An unbalanced branch and bound tree</td>
<td>30</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Example of cyclic flow forming an origin-destination path</td>
<td>38</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Topology of problem instances</td>
<td>47</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Unbalanced and balanced node selection</td>
<td>54</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Toronto: Single Variable</td>
<td>55</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Toronto: Node Connection Cover</td>
<td>55</td>
</tr>
<tr>
<td>7.5.4</td>
<td>Toronto: Balanced $st$-cut</td>
<td>56</td>
</tr>
<tr>
<td>7.5.5</td>
<td>Toronto: Shortest Path $st$-cut</td>
<td>56</td>
</tr>
<tr>
<td>7.5.6</td>
<td>Toronto: Single Path</td>
<td>57</td>
</tr>
</tbody>
</table>
7.5.7 Toronto: Congested Arc Rule 57
7.5.8 Toronto: CPLEX Default MIP Solver 58
7.5.9 bhv5: Single Variable 58
7.5.10 bhv5: Node Connection Cover 59
7.5.11 bhv5: Balanced $st$-cut 59
7.5.12 bhv5: Shortest Path $st$-cut 60
7.5.13 bhv5: Single Path 60
7.5.14 bhv5: Congested Arc Rule 61
7.5.15 bhv5: CPLEX Default MIP Solver 61

8.3.1 Pricing subproblems for a single branching constraint of the single path rule 70
8.3.2 Pricing subproblems for multiple branching constraints of the single path rule 71
CHAPTER 1

Introduction

Communication industries play a pivotal role in today’s society. These industries manage vast amounts of data across increasingly complex networks. The ability to efficiently administer this data poses a difficult problem in the design and management of networks.

Certain types of communication, such as video conferencing, require that an origin-destination demand be routed on a single path. In other words, the data must stay together as an entire unit as it is routed from its source to its destination. In the situation of a two-way video conferencing between companies A and B, two origin-destination demands would have to be satisfied: a demand from A to B and a demand from B to A. Telecommunication industries often have to route numerous of these origin-destination demands simultaneously. A problem based on this kind of network is known as an origin-destination integer multi-commodity flow problem (ODIMCFP).

Telecommunication industries are often expected to provide their customers with a certain quality of service. A common measure of network usage is the maximum link congestion of a given routing, i.e., the maximum percentage of used link capacity. As the maximum link congestion increases, the network becomes more prone to instability in the event of a change in demands. For example, a sporting event or other large gathering would change the local demands for cell phone traffic, resulting in a more congested network and likely more dropped calls. In order to maintain a high quality of service, a telecommunications industry may try to limit the maximum amount of congestion that can appear on any link in the network.
Often times the demands on a communication network are not known a priori. For our purposes, we are assuming that all demands are known beforehand. This is a likely scenario in backbone networks, where large demands can be expected to be constant throughout network operation.

Our problem is a minimum congestion ODIMCFP. For our problem, each demand has to be satisfied along a single path from origin to destination. The objective is to find a routing that satisfies all demands and minimizes the maximum link congestion of the network.

The single path requirement of our problem creates a difficult mixed integer programming (MIP) problem. A common technique used to solve an MIP is branch and bound. Branch and bound begins from a relaxation of the MIP that, in our case, allows the flow of data to follow multiple paths, i.e., allowing the blocks of data to break up. A branching rule is then used to remove infeasible solutions. An efficient branch and bound algorithm can greatly reduce the amount of time needed to find an optimal solution.

In this paper, we present numerous branching rules to solve the minimum congestion ODIMCFP. In the next chapter we present introductory information on the concepts used for creating our model and branching rules.
CHAPTER 2

Basic Concepts

2.1. Concepts of Graph Theory

A graph is a collection of nodes (or vertices) and node pairs called edges. Each edge in the graph can be directed or undirected, weighted or unweighted. Network flow problems are defined on graphs and generally involve flow traveling across the arcs.

![Small directed graph](image)

**Figure 2.1.1.** Small directed graph

A directed graph $G = (N, A)$ is a collection of nodes and directed edges, known as arcs. Figure 2.1.1 depicts a directed graph where $N = \{1, 2, 3, 4\}$ is the node set and $A = \{(1, 2), (1, 3), (2, 3), (3, 4), (4, 1)\}$ is the set of arcs. Each arc $(i, j) \in A$ refers to a connection from node $i$ traveling to node $j$, where $i$ is known as the tail node and $j$ is known as the head node. A directed path is a sequence of alternating nodes and arcs, starting and ending with a node, in which no node is repeated and for each arc, the tail node matches the arc’s predecessor node (on the path) and the head node matches the arc’s successor node. In the graph $G$, $1 \rightarrow (1, 2) \rightarrow (2, 3) \rightarrow (3, 4)$ is a path. The path can be represented in arc formulation as $(1, 2) - (2, 3) - (3, 4)$ or in node formulation as $1 \rightarrow 2 \rightarrow 3$. A directed cycle is a directed path with an additional arc connecting the
last node to the first. In the graph $G$, $1 - (1, 2) - 2 - (2, 3) - 3 - (3, 4) - 4 - (4, 1)$ is a cycle. Figure 2.1.2 depicts these concepts.

![Figure 2.1.2. Illustrations of graph definitions](image)

A *tree* is a connected graph which contains no cycles. For the purposes of this paper, we will be dealing with a *directed-out-tree*. Trees are often defined in terms of predecessor and successor relationships. The root node of a tree is a specially designated node that has no predecessors. A tree is a directed-out-tree rooted at node 1 if there is a directed path from node 1 to every other node in the tree; see Figure 2.1.3. A technique called branch and bound, which will be discussed in detail later, makes extensive use of a directed-out-tree to keep track of subproblems.

![Figure 2.1.3. A directed-out-tree](image)

Further readings on graph theory can be found in the book by Berge [5].

### 2.2. Network Flow Problems

In this section, we mention three types of problems: a single commodity flow problem, a multi-commodity flow problem (MCFP) and an origin-destination integer multi-commodity flow problem (ODIMCFP).
2.2.1. Minimum Cost Flow Model. Minimum cost flow models are probably the simplest examples of network flow problems. The minimum cost flow problem consists of minimizing the cost of shipping a commodity through a directed graph in order to satisfy a set of supplies and demands. Each arc in the graph has a cost (per unit of flow) and a capacity associated with it.

Let $G = (N, A)$ be a directed graph defined by a set $N$ of $n$ nodes and a set $A$ of $m$ directed arcs. Each arc $(i, j) \in A$ has a parameter $c_{ij}$ denoting the cost per unit of flow, a maximum capacity $u_{ij}$ and a minimum capacity $\ell_{ij}$. For each node $i \in N$, let $b(i)$ denote the supply/demand of that node, where: if $b(i) > 0$ then node $i$ is a supply node, if $b(i) < 0$ then node $i$ is a demand node and if $b(i) = 0$ then node $i$ has no supply or demand. The minimum cost flow problem can be modeled as follows:

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

s.t. $\sum_{j: (i,j) \in A} x_{ij} - \sum_{j: (j,i) \in A} x_{ji} = b(i) \quad \forall i \in N$ \hspace{1cm} (1)

$$\ell_{ij} \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A \hspace{1cm} (2)$$

The decision variables in the problem are the $x$ variables. These decision variables determine the total amount of flow through each arc. The first set of constraints is known as the flow conservation constraints, where the first term of the constraint, $\sum_{j: (i,j) \in A} x_{ij}$, denotes the total out flow of node $i$, and the second term, $\sum_{j: (j,i) \in A} x_{ji}$, denotes the total inflow of node $i$. The flow conservation constraint states that the total outflow of node $i$ minus the total inflow of node $i$ must equal the supply/demand of node $i$. The second set of constraints is known as the capacity constraints. These constraints ensure that the total flow through an arc is within proper lower and upper bounds of the arc. The problem can be rewritten in matrix form as:
\[
\begin{align*}
\min & \quad c'x \\
\text{s.t.} & \quad Ax = b \\
& \quad \ell \leq x \leq u.
\end{align*}
\]

The first constraint corresponds to the flow conservation constraints. Here, \( A \) is an \( n \times m \) matrix known as the node-arc incidence matrix or more concisely the incidence matrix. Each column of \( A \) refers to an arc \((i, j) \in A\) and each row of \( A \) refers to a node \( i \in N \). The column referring to an arc \((i, j) \in A\) has a one in the row corresponding to node \( i \), a negative one in the row corresponding to node \( j \) and zeros in every other row of that column. Let \( A_{i \bullet} \) denote the row vector corresponding to node \( i \in N \). Then the dot product of \( A_{i \bullet} \) and \( x \) gives the left side of the flow conservation constraint for node \( i \), i.e., \( A_{i \bullet} \cdot x = \sum_{j: (i,j) \in A} x_{ij} - \sum_{j: (j,i) \in A} x_{ji} \).

The \( m \)-vectors \( c, \ell \) and \( u \) correspond to the cost, lower and upper bounds of an arc, respectively. For most problems, the lower bound of each decision variable \( x \) is zero and the \( \ell \) is represented by non-negativity.

\section*{2.2.2. Multi-Commodity Flow Problems.} For a multi-commodity flow problem (MCFP) there are different types of flow that must travel through a network and meet a set of supplies and demands. Consider the graph \( G = (N, A) \) and a set of commodities \( K \), then for \( i \in N \) the supply or demand of commodity \( k \in K \) at node \( i \) can be denoted \( b_k(i) \). Similarly for every \((i, j) \in A\) and every \( k \in K \) there is a flow variable \( x_{ij}^k \) that refers to the amount of flow from commodity \( k \) that passes through arc \((i, j)\). The capacity constraints are satisfied if for each arc the sum of flows over all commodities is within the arc bounds, i.e., \( \ell_{ij} \leq \sum_{k \in K} x_{ij}^k \leq u_{ij}, \forall (i,j) \in A \). Letting \( x^k \) denote the vector of all flow variables for commodity \( k \) the problem can be represented in matrix form as:
\[
\min \sum_{k \in K} c^k x^k \\
\text{s.t. } Ax^k = b_k \quad \forall k \in K \\
\ell \leq \sum_{k \in K} x^k \leq u.
\]

**Example 1:** Consider the small network with two commodities \(D\) and \(B\) in Figure 2.2.1. Commodity \(D\) has to send three units of flow from node 1 to node 3. Commodity \(B\) has to send two units of flow from node 2 to node 4. Each arc in Figure 2.2.1 is labeled with \((\text{cost, capacity})\) denoting the cost per unit of flow and the maximum capacity of the arc. Each arc has a minimum capacity of zero.

![Figure 2.2.1. A small example of a multi-commodity flow problem](image)

The problem can be represented in node-arc formulation as:
The first ten constraints are the flow conservation constraints and the last five ones are the capacity constraints. Notice that the flow conservation constraints can be partitioned into two sets $Ax^1 = b_1$ and $Ax^2 = b_2$ where $A$ is the node arc incidence matrix.

$$A = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 \\
0 & -1 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \quad b_1 = \begin{pmatrix}
3 \\
-3 \\
0 \\
0 \\
0 \\
\end{pmatrix} \quad b_2 = \begin{pmatrix}
0 \\
2 \\
0 \\
-2 \\
0 \\
\end{pmatrix}$$

An optimal solution of the problem is $x^1_{1,2} = x^2_{2,3} = 3$, $x^2_{2,3} = x^2_{2,4} = x^2_{3,4} = 1$ with an optimal objective value of 19. Note that commodity $D$ sends its entire flow through the path $1 \rightarrow 2 \rightarrow 3$ and commodity $B$ sends its flow through two paths, half through the path $2 \rightarrow 3 \rightarrow 4$ and half through the path $2 \rightarrow 4$.

### 2.2.3. Origin-Destination Integer Multi-Commodity Flow Problem

The origin-destination integer multi-commodity flow problem (ODIMCFP) is a multi-commodity flow problem that requires each commodity to travel along a single acyclic path from origin to destination.

Consider an ODIMCFP with a directed graph $G = (N, A)$ and a set of commodities $K$. For each commodity $k \in K$ let $(s_k, t_k)$ represent the origin and destination nodes and let $v_k$ represent the volume of traffic for commodity $k$. A common technique for modeling this type of problem is as a 0-1 integer programming problem. That is, we can formulate the model with a set of decision variables $f^k_{ij} \in \{0,1\}$: $k \in K$, $(i,j) \in A$, where $f^k_{ij} = 1$ if the entire amount of flow of commodity $k \in K$ passes through arc $(i,j)$ and is zero otherwise. Each decision variable of the MCFP would be transformed such that $x^k_{ij} = v_k f^k_{ij}$. Letting $c_{ij}$ and $u_{ij}$ denote the cost and capacity of arc $(i,j) \in A$, the node-arc formulation of the ODIMCFP becomes:
\[
\begin{align*}
\min & \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} v_k f^k_{ij} \\
\text{s.t.} & \quad \sum_{j: (i,j) \in A} f^k_{ij} - \sum_{j: (j,i) \in A} f^k_{ji} = \begin{cases}
1 & \text{if } i = s_k \\
-1 & \text{if } i = t_k \\
0 & \text{otherwise}
\end{cases} \quad \forall i \in N, \forall k \in K \\
& \quad \sum_{k \in K} v_k f^k_{ij} \leq u_{ij} \quad \forall (i,j) \in A \\
& \quad f^k_{ij} \in \{0, 1\} \quad \forall (i,j) \in A, \forall k \in K.
\end{align*}
\]

An example of an ODIMCFP can be found in the branch and bound section on page 13. Further information on network flow problems can be found in Ahuja et al. [1].

### 2.3. Computational Complexity

The complexity of a class of optimization problems determines the amount of effort a computer may have to spend to solve a problem and verify that a given solution is optimal. This can often be thought of as the worst case amount of time to solve a problem, and the worst case amount of time to verify that a given solution is optimal as a function of the problem’s input length. A problem that can be solved with a polynomial algorithm is usually referred to as easy because the amount of time to solve the problem will be bounded by a polynomial function of input length. In optimization, many problems cannot currently be solved with a polynomial algorithm, i.e., the amount of time needed could grow exponentially with the size of the instance. Problems for which no polynomial algorithm is known are considered hard.

A problem is said to be in the class \( \mathcal{P} \) if there exists a polynomial time algorithm to solve every possible instance of that problem. All linear programs exist in the class \( \mathcal{P} \) and can thus be considered easy.
The class $\mathcal{NP}$ refers to a set of decision problems, i.e., problems that require an answer that is “Yes” or “No”, such that any given solution can be checked for optimality in polynomial time. It can be easily shown that $\mathcal{P} \subseteq \mathcal{NP}$ because any given solution of a problem in $\mathcal{P}$ can be checked for optimality in polynomial time by ignoring the given solution and simply solving the problem.

The class of $\mathcal{NP}$-complete problems refers to a set of decision problems that are polynomial time reducible to each other and considered the hardest problems, in terms of complexity, in $\mathcal{NP}$. There does not currently exist a polynomial algorithm to solve an $\mathcal{NP}$-complete problem and the result of such an algorithm would mean that every problem in the class $\mathcal{NP}$ could be solved with a polynomial time algorithm and thus $\mathcal{NP} = \mathcal{P}$. It is believed by many that no such polynomial time algorithm exists but this has yet to be verified.

A problem $\Pi$ is considered $\mathcal{NP}$-hard if a polynomial algorithm to solve $\Pi$ implies $\mathcal{NP} = \mathcal{P}$. However, if $\mathcal{NP} = \mathcal{P}$, this would not imply with certainty that $\Pi$ can be solved with a polynomial time algorithm. A common method to demonstrate that a problem $\Pi$ is $\mathcal{NP}$-hard is by showing that any possible instance of an $\mathcal{NP}$-complete problem $\Lambda$ could be transformed, in polynomial time, to an instance of the problem $\Pi$ such that a solution of the instance of $\Pi$ would give a correct solution to the instance of $\Lambda$. As a result, if there exists a polynomial time algorithm for $\Pi$, then there would exist a polynomial time algorithm for the $\mathcal{NP}$-complete problem $\Lambda$ and thus $\mathcal{NP} = \mathcal{P}$.

A problem $\Pi$ is considered $\mathcal{NP}$-easy if any solution $\bar{x}$ along with a claim $\tau$ can be verified in polynomial time. In other words, there exists a polynomial time algorithm that proves that $\bar{x}$ is feasible to $\Pi$ and that the claim $\tau$ is true. A problem that is both $\mathcal{NP}$-easy and $\mathcal{NP}$-hard is equivalent in difficulty to $\mathcal{NP}$-complete, i.e., there exists a polynomial time algorithm if and only if $\mathcal{NP} = \mathcal{P}$. As we will demonstrate in Section 3.2, the minimum congestion ODIMCFP is both $\mathcal{NP}$-hard and $\mathcal{NP}$-easy.
For further reading on computational complexity, we refer the reader to Garey and Johnson [7].

2.4. Branch and Bound

As will be made clear below, the problem we consider is a mixed integer programming (MIP) problem. The branch and bound technique is a common approach to solving a MIP problem. The process works by finding the solution of the LP relaxation of the original problem. This is obtained by eliminating the integrality constraints on the variables. This creates a linear program, which contains, among others, all feasible solutions of the MIP. If the solution of the LP is not integer feasible, then branching occurs, and the MIP is partitioned into subproblems. Each subproblem has an added constraint that makes the current LP solution infeasible to the relaxation of the subproblem. Collectively, all feasible solutions of the MIP are preserved in the subproblems. The branch and bound process is then repeated for the subproblems. Solutions of the subproblems give upper and lower bounds on the MIP. The upper bound refers to the minimum objective value of all integer feasible subproblems. If a subproblem has a non-integer solution, the objective value gives a lower bound for that subproblem, i.e., an integer feasible solution of the subproblem cannot have a better objective value than its lower bound. If the lower bound solution is greater than or equal to the upper bound, the subproblem can be removed from the branch and bound tree. This is called as bounding.

The subproblems of the branch and bound process create a directed-out-tree rooted at the original MIP problem, and the nodes of the tree represent each subproblem. Once all subproblems of the branch and bound tree are either linear infeasible, integer feasible, or removed by bounding, the process ends and an optimal solution is found.

Consider an ODIMCFP $P$ over a graph $G = (N, A)$ with a set of demands $K$ and let $x$ denote the flow variables of $P$. The optimal objective value $z$ of the problem
\( P \) can be found using branch and bound. A sketch of a generic branch and bound algorithm is given below:

- \( \mathcal{L} = \{ P \} \)
- \( z_{ub} = \infty \) (The best found solution of \( P \))
- \( w = 0 \)
- \( \textbf{while } \mathcal{L} \neq \emptyset \)
  - select \( P_i \) from \( \mathcal{L} \); \( (P_i \) is any problem in \( \mathcal{L} \))
  - \( \mathcal{L} \leftarrow \mathcal{L} \setminus \{ P_i \} \)
  - create an LP relaxation \( \tilde{P}_i \) from \( P_i \)
  - solve \( \tilde{P}_i \rightarrow x^*, z_{lb} \)
  - \textbf{if } \( x^* \) is not feasible for \( P_i \)
    - \textbf{if } \( z_{lb} < z_{ub} \)
      - select \( j : x_j^* \not\in \mathbb{Z} \)
      - \( P_{w+1} = \{ P_i : x_j \leq \lfloor x_j^* \rfloor \} \)
      - \( P_{w+2} = \{ P_i : x_j \geq \lceil x_j^* \rceil \} \)
      - \( \mathcal{L} \leftarrow \mathcal{L} \cup \{ P_{w+1}, P_{w+2} \} \)
      - \( w \leftarrow w + 2 \)
  - \textbf{else if } \( z_{lb} < z_{ub} \)
    - \* \( z_{ub} \leftarrow z_{lb} \)
  - \* \( z \leftarrow z_{ub} \).

Consider the ODIMCFP in Figure 2.4.1. Commodity \( D \) must send three units from node 1 to node 3 and commodity \( B \) must send two units from node 2 to node 4.

Let \( P \) denote the ODIMCFP model of Figure 2.4.1 having decision variables \( f_{ij}^k \in \{0, 1\} \). The relaxation \( \tilde{P} \) of \( P \) would be the same model except with \( f_{ij}^k \in [0, 1] \). The problem \( \tilde{P} \) can then be solved giving the solution:

\[ z_{lb} = 13; \]
\[ f_{1,2} = f_{2,3} = 1, \quad f_{2,3} = f_{2,4} = f_{3,4} = 0.5; \]
\[ f_{1,5} = f_{2,4} = f_{3,4} = f_{5,4} = f_{1,3} = f_{1,5} = f_{5,4} = 0. \]

This optimal solution of \( \tilde{P} \) is infeasible to \( P \) since \( f_{2,3}^2, f_{2,4}^2, f_{3,4}^2 \notin \{0, 1\} \), so branching must occur on one of those variables. Selecting \( f_{2,3}^2 \) to branch on, we can create the two subproblems \( P_1 = \{ P : f_{23}^2 = 0 \} \) and \( P_2 = \{ P : f_{23}^2 = 1 \} \). The relaxation \( \tilde{P}_1 \) of \( P_1 \) is then solved giving the solution:

\[ z_{lb} = 16; \]
\[ f_{1,2} = f_{2,3}^1 = f_{2,4}^1 = 1; \]
\[ f_{1,5} = f_{2,4} = f_{3,4} = f_{5,4} = f_{1,3} = f_{1,5} = f_{2,3} = f_{3,4} = f_{5,4} = 0. \]

The optimal solution of \( \tilde{P}_1 \) is feasible to \( P \) since all flow variables are either zero or one. This is the best solution found for \( P \) so \( z_{ub} = 16 \). The relaxation \( \tilde{P}_2 \) of \( P_2 \) is then solved but this problem is infeasible to the LP. All subproblems have been solved, so the branch and bound algorithm terminates giving the optimal solution to \( P \):

\[ z = 16; \]
\[ f_{1,2} = f_{2,3} = f_{2,4} = 1; \]
\[ f_{1,5} = f_{2,4} = f_{3,4} = f_{5,4} = f_{1,3} = f_{1,5} = f_{2,3} = f_{3,4} = f_{5,4} = 0. \]
CHAPTER 3

Minimum Congestion Model

3.1. Model

The minimum congestion routing variant of the ODIMCFP is a well known variation of the minimum cost flow problem. This problem involves finding an integer feasible solution of an ODIMCFP which minimizes the maximum congestion in the network.

We consider this problem over a directed graph $G = (N, A)$ with a set of commodities $K$. For every commodity $k \in K$ there is a node pair $(s_k, t_k) \in N$ denoting the origin and destination node (also called source and sink nodes) with a supply or demand of $v_k$ for commodity $k$. Note that for our data sets, every arc $(i, j) \in A$ is coupled with a reverse arc $(j, i) \in A$ along the same edge.

The congestion ratio is a well known measure for routing problems in telecommunications. This measure refers to the percentage of used arc capacity of a given arc and routing. The maximum congestion ratio of a routing refers to the congestion ratio of the arc with the most congestion. In other words, the congestion of every arc in the network is less than or equal to the maximum congestion ratio for a given routing. Thus the minimum congestion of a network can be represented as a min-max problem, i.e., determining a routing that minimizes the maximum congestion ratio of all feasible routings.

For our model, we minimize a decision variable $\alpha$ representing the maximum congestion ratio of a routing. There is no upper bound on $\alpha$ and it is coupled with the capacity $u_{ij} : (i, j) \in A$ for every capacity constraint in the model, i.e., for each
of the capacity constraints, the corresponding capacity $u_{ij}$ is replaced with $\alpha u_{ij}$. This creates an uncapacitated min-max problem, as there is no upper bound on the amount of flow that may pass through a single arc. Thus, if the optimal solution of the problem has an objective greater than one the problem would be infeasible with the traditional $u_{ij}$ capacity constraints. This type of scenario could be used to determine if a communications network could withstand a sudden influx of demands in the network.

As discussed earlier, the ODIMCFP requires a single acyclic path to send a flow of $v_k$ from $s_k$ to $t_k$ for every $k \in K$. The decision variables are $f_{ij}^k \in \{0, 1\} : k \in K, (i, j) \in A$, where $f_{ij}^k = 1$ if the entire amount of demand of commodity $k \in K$ passes through arc $(i, j)$ and is zero otherwise. We refer to the decision variables $f_{ij}^k$ throughout this paper as flow variables. Putting all this together, we form the compact formulation of the minimum congestion ODIMCFP.

\[
\begin{align*}
\min & \quad \alpha \\
\text{s.t.} & \quad \sum_{j: (i,j) \in A} f_{ij}^k - \sum_{j: (j,i) \in A} f_{ji}^k = \\
& \quad \begin{cases} 
1 & \text{if } i = s_k \\
-1 & \text{if } i = t_k \\
0 & \text{otherwise}
\end{cases} \quad \forall i \in N, \forall k \in K \\
& \quad \sum_{k \in K} v_k f_{ij}^k - \alpha u_{ij} \leq 0 \quad \forall (i, j) \in A \\
& \quad f_{ij}^k \in \{0, 1\} \quad \forall (i, j) \in A, \forall k \in K.
\end{align*}
\]

We use the following notation:

$G = (N, A)$: the graph $G$ with nodes $N$ and arcs $A$;

$u_{ij}$: the capacity of arc $(i, j)$;

$K$: the set of all commodities in the network;

$v_k$: the amount of flow being supplied/demanded for commodity $k$;
\( \alpha \): the decision variable referring to the maximum congestion ratio of all arcs in a routing;
\[ f_{ij}^k \in \{0, 1\}, \ \forall (i, j) \in A, \ \forall k \in K: \text{decision variables that equal one if the entire amount of commodity } k \text{ passes through arc } (i, j) \text{ and zero otherwise.} \]

To solve this model, we relax the integrality constraints on flow variables so that they can take on real values. We then employ a branching rule to partition the feasible set and to eliminate infeasible solutions. Thus, for the LP relaxation we have:

\[ f_{ij}^k \in [0, 1], \ \forall (i, j) \in A, \ \forall k \in K. \]

As we will show, this model presents numerous difficulties for developing and implementing branching rules. We explored several ways to overcome these difficulties. In the future, we hope to expand this model to a column generation format and apply our branching rules using \textit{branch and price}. This should yield a more efficient solver for the problem.

### 3.2. Complexity of the Problem

The minimum congestion ODIMCFP is \( \mathcal{NP} \)-hard. We demonstrate this by showing that every instance of the \textit{multiprocessor scheduling problem} (a well known strongly \( \mathcal{NP} \)-complete problem) can be transformed to an instance of our model, such that the solution of our model would give a correct “yes” or “no” answer to the multiprocessor scheduling problem.

The multiprocessor scheduling problem has a set of tasks \( T \), a length of time for each task \( l(t) \in \mathbb{Z}^+ \), \( \forall t \in T \), a number of processors \( m \in \mathbb{Z}^+ \) used to complete the task, and a deadline \( D \in \mathbb{Z}^+ \). The problem determines whether or not a set of tasks can be processed on the set of processors by the given deadline. The answer is “yes” if the deadline can be met and “no” if it cannot [7].
Any instance of the multiprocessor scheduling problem with \( m \) processors, a set of tasks \( T \), a length of task \( l(t) \), \( t \in T \) and a deadline \( D \) can be transformed into an instance of the minimum congestion ODIMCFP with graph \( G = (N, A) \) and set of commodities \( K \). Figure 3.2.1 depicts the structure of the graph \( G \). The graph contains \( 2 \times m \) arcs and \( m + 2 \) nodes. For every \( a \in A \) the capacity is \( D \). Every \( k \in T \) corresponds to a commodity \( k \in K \) with origin-destination pair \((s, t)\) (as depicted in Figure 3.2.1) with supply/demand of \( v_k = l(k) \).

![Figure 3.2.1. Graphing a multi-processor problem](image)

The optimal solution of this problem gives an ordering of tasks that minimizes the times spent processing. If commodity \( k \) passes through arcs \((s, i) \) and \((i, t)\), \( i = 1, 2, \ldots, m \), then processor \( i \) is used for task \( k \). The optimal objective corresponds to the percentage of time \( D \) needed to complete the tasks. If the objective function is greater than one, then the deadline cannot be met and the answer to the multiprocessor problem is “no”. If the objective function is less than or equal to one, then there is sufficient time, and the answer to the problem is “yes”. This transformation of an \( \mathcal{NP} \)-complete problem into a minimum congestion ODIMCFP demonstrates that our problem is at least as difficult as an \( \mathcal{NP} \)-complete problem and completes our proof that our problem is \( \mathcal{NP} \)-hard.
In addition, the minimum congestion ODIMCFP is \(NP\)-easy, i.e., given a feasible solution and claim, it can be verified in polynomial time that the given solution is feasible and the claim is true. Consider the given solution \((\bar{f}, \bar{\alpha})\) and the claim that the minimum congestion is less than or equal to \(\tau\). A simple algorithm to authenticate the given solution and claim would be:

- if \(\bar{\alpha} > \tau\), the given solution does not support the claim;
- scan \(\bar{f}\) one time for each constraint in the model;
  - if the constraint is not satisfied, the solution is not feasible.

The size of \(\bar{f}\) and the number of times to scan \(\bar{f}\) are both bounded by the size of the model. This means that any given solution \((\bar{f}, \bar{\alpha})\) and claim \(\tau\) could be verified in polynomial time. Thus the minimum congestion ODIMCFP is both \(NP\)-easy and \(NP\)-hard.

### 3.3. Complexity for a Single Commodity

The minimum congestion ODIMCFP for a single commodity is in \(P\). Consider the graph \(G = (N, A)\) with capacity \(u_{ij}\) for all \((i, j) \in A\) and a single commodity \(k\) with origin-destination nodes \((s_k, t_k)\) and a demand of traffic \(v_k\). A feasible solution would consist of a single path \(p\) with flow from \(s_k \rightarrow t_k\). The congestion of all arcs not in \(p\) would be zero and the congestion of an arc \((i, j) \in p\) would be \(\frac{v_k}{u_{ij}}\). The most congested arc in the path \(p\) would be the arc with minimum capacity. We can define the capacity of a path \(p\) as the capacity of the arc with minimum capacity in \(p\). The larger the capacity of \(p\), the less congested the solution would be. A feasible path with maximum capacity would provide a solution with minimum congestion. As a result, the minimum congestion ODIMCFP for a single commodity can be solved as a maximum capacity path problem.

The maximum capacity path problem can be solved with a slightly modified version of Dijkstra’s algorithm [1]. When implemented using arrays, this is a polynomial
algorithm that has an overall complexity of $O(|N|^2)$, i.e., the maximum run time is bounded by a function of the number of nodes squared. The minimum congestion ODIMCFP for a single commodity can be solved with this algorithm and is thus in $P$.

### 3.4. Difficulties with the Model

It is often better to model large, specially structured problems, such as a multi-commodity flow problem, with fewer constraints even though this might imply a very large number of decision variables [2]. We will discuss in detail how our model can be transformed from its compact formulation into a path model in Chapter 8. For the compact formulation, we discovered numerous difficulties with our model.

During the LP phase, most of the work associated with solving the LP is used in updating and storing the inverse basis matrix of the model. The dimension of the basis corresponds to the number of constraints in the model. The larger the basis, the more work the LP solver has to do. For this reason, a model with fewer constraints could greatly improve the efficiency of the LP phase.

Symmetry in the model was also a problem. Symmetry occurs when very similar infeasible solutions continuously appear in the branch and bound tree. This results in numerous redundant branches. For our problem, the compact formulation can greatly increase the effects of symmetry because of cyclic flow.

During the LP phase of the problem, it was possible and often the case that CPLEX would return a solution with flow passing through arcs that satisfy the flow conservation constraints but do not travel from a source node to a destination node. Cyclic flow occurs when multiple arcs are not fully saturated from the commodity demands and a cycle of flow forms. Since the arcs were not fully saturated, the cycle of flow would have no impact on the objective function. Thus CPLEX would return
a solution that is optimal to the LP but not ideal for developing branching rules and solving the MIP.

Cyclic flow can also result in many redundant branches during the branching process. It was often possible for branching rules to be satisfied by cyclic flow, and not from the intended flow caused by the demands. This is problematic because it would result in many redundant subproblems, which would greatly increase the size of the branch and bound tree.

One of the major problems we faced early on was that non-integer cyclic flow would result in an integer optimal solution of the LP that was not recognized by the optimization software CPLEX. In other words, there would exist a path with integer feasible flow for each commodity but, because of a cycle of fractional flow, CPLEX would not recognize the solution as integer feasible.

Figure 3.4.1 depicts the possible flow values of a LP solution for a commodity with source-destination pair (1, 5). Clearly, path 1 – 2 – 3 – 5 is a feasible integer path for the commodity. If every other commodity of the LP solution had an integer feasible path from source to destination, CPLEX would not recognize the solution as optimal because of the infeasible cyclic flow across the edges (1,3) and (6,4). This type of scenario was remedied by designing an algorithm to recognize integer feasible solutions and inject the solution into CPLEX.
CHAPTER 4

Contributions

We present six variants of branch and bound algorithms that differ in their branching rules. Two branching rules are from the literature [3] and we introduce four of our own. We provide a theoretical analysis of each of the six branching rules. We present the computational results of these branching rules, along with the results of the CPLEX default MIP solver for a wide array of problem instances. For a selected set of instances we provide graphical imagery of the branch and bound tree after a limited time period for each of our rules.

In Chapter 8, we present two methods of implementing our branching rules using a path formulation and branch and price. One method implements the branching rule by adding constraints to the pricing problem, and the other method by adding constraints to the restricted master problem.

In branch and price, careful consideration has to be taken when adding constraints to the pricing problem. It is important that the rule does not destroy the structure of the pricing problem and that the time it takes to solve the pricing problem remains tractable throughout the branch and price process. Both branching rules from the literature retain the structure of the pricing problem, but only one of the rules remains tractable (this was intended and highlights the importance of tractability). The tractable method is a rule which generates two problems with common solutions at each branch and thus not a partition, i.e., a feasible solution can exist in both branches. We present a new branching rule which provides a partition of the feasible set and can be applied to the pricing problem without destroying its structure or becoming intractable.
The two branching rules from the literature resulted in “problem symmetry”, i.e., there were numerous redundant branches when applied to branch and price [3]. The authors remedied this problem by using heuristic methods to apply cuts after the pricing phase, resulting in a branch-and-price-and-cut implementation. We devised a branching method that performed well using only branch and bound. For its implementation in branch and price, this method could not be used by adding constraints to the pricing problem because it would destroy its structure. However, we demonstrate how this rule can be applied to branch and price by transforming the branching constraint into terms that contain path variables and adding the constraint to the restricted master problem. We then provide a method for calculating the reduced costs of the path variables.
CHAPTER 5

Developing Branching Rules

5.1. Constructing a Branching Rule

During the process of developing branching algorithms, we discovered multiple properties regarding an ODIMCFP. From these properties we were able to develop theorems that could be extended to create a wide variety of branching rules.

Consider the ODIMCFP with graph $G = (N, A)$ and a set of commodities $K$. For each commodity $k \in K$, there exists at least one $st$-cut. An $st$-cut for a commodity $k$ with source node $s_k$ and sink $t_k$ is a partition of all nodes in the graph into two sets $S$ and $\bar{S} = N \setminus S$ such that $s_k \in S$ and $t_k \in \bar{S}$. For any $st$-cut $S$ and $\bar{S}$, there exists a set of arcs $\delta(S)$ consisting of all arcs that have their tail node in $S$ and head node in $\bar{S}$ and a set $\delta(\bar{S})$ consisting of all arcs that have their tail node in $\bar{S}$ and head node in $S$, i.e.:

$$\delta(S) = \{(i, j) \in A : i \in S, j \in \bar{S}\}$$

$$\delta(\bar{S}) = \{(i, j) \in A : i \in \bar{S}, j \in S\}.$$

Let $F$ denote the sum of all flow variables of commodity $k$ through the arcs in $\delta(S)$. Similarly, we let $B$ denote the sum of all flow variables through $\delta(\bar{S})$:

$$F = \sum_{(i,j) \in \delta(S)} f_{ij}^k$$
\[ B = \sum_{(i,j) \in \delta(S)} f_{ij}^k. \]

From our model, we know that the total supply of commodity \( k \) on the source side of the cut is one and the total demand on the supply side of the cut is one. Thus the amount of flow of \( F \) must be exactly one more than the total amount of flow in \( B \).

**Claim 1.** For any \( st \)-cut with a supply and demand of 1, if \( F \) denotes the total flow across the cut from \( s \) to \( t \) and \( B \) denotes total flow across the cut from \( t \) to \( s \), then

\[ F - B = 1. \]

Next, we can partition \( \delta(S) \) into two sets \( \delta_1(S) \) and \( \delta_2(S) \) such that \( \delta(S) = \delta_1(S) \cup \delta_2(S) \) and \( \delta_1(S) \cap \delta_2(S) = \emptyset \) and define the variables \( F_1 \) and \( F_2 \) as:

\[ F_1 = \sum_{(i,j) \in \delta_1(S)} f_{ij}^k \]

\[ F_2 = \sum_{(i,j) \in \delta_2(S)} f_{ij}^k. \]

We define \( \delta_1(S) \), \( \delta_2(S) \) and \( B_1 \), \( B_2 \) in the same fashion:

\[ B_1 = \sum_{(i,j) \in \delta_1(S)} f_{ij}^k \]

\[ B_2 = \sum_{(i,j) \in \delta_2(S)} f_{ij}^k. \]

Figure 5.1.1 depicts a possible origin-destination path of commodity \( k \) along with a partition of \( st \)-cut arcs. Let the darker shaded region on top contain flow variables
from $F_1$ and $B_1$ and let the lighter shaded region on bottom contain the flow variables from $F_2$ and $B_2$. With this partition, we discuss below about the feasible set of solutions of an ODIMCFP.

**Theorem 1.** For any st-cut and any partition of the st-cut arcs, a feasible solution of the ODIMCFP will fall into exactly one of the two possible situations

$$F_1 - B_1 - (F_2 - B_2) \geq 1 \quad \text{or} \quad F_1 - B_1 - (F_2 - B_2) \leq -1.$$

**Proof.** Each of the variables $F_1$, $F_2$, $B_1$, $B_2$ are sums of $\{0,1\}$ variables and thus the left-hand side of the inequalities exists in $\mathbb{Z}$. Hence it suffices to show that $F_1 - B_1 - (F_2 - B_2) \neq 0$ for any possible partition.

$$F_1 - B_1 - (F_2 - B_2) = F_1 - F_2 - B_1 + B_2 + 1 - 1$$

$$= F_1 - F_2 - B_1 + B_2 + (F - B) - 1 \quad \text{(by Claim 5.1.1)}$$

$$= F_1 - F_2 - B_1 + B_2 + (F_1 + F_2) - (B_1 + B_2) - 1$$

$$= 2(F_1 - B_1) - 1.$$
2(F_1 - B_1) - 1 must be an odd number, so it cannot possibly equal zero. Thus one and only one of the above two situations will be true. □

Note that the proof of this theorem also allows us to rewrite the conditions solely in terms of F_1 and B_1, i.e., F_1 - B_1 - (F_2 - B_2) = 2(F_1 - B_1) - 1. This implies that F_1 - B_1 ≥ 1 and F_1 - B_1 ≤ 0 are equivalent expressions for F_1 - B_1 - (F_2 - B_2) ≥ 1 and F_1 - B_1 - (F_2 - B_2) ≤ -1 respectively.

This is an important property for the early formulation of our branching rules. The result is that the feasible set of solutions of an ODIMCFP can be partitioned into two groups: the group with F_1 - B_1 ≤ 0 and the group with F_1 - B_1 ≥ 1. Thus, this property can be implemented as a branching rule if there exists a set of cut arcs in the LP phase such that 0 < F_1 - B_1 < 1. We will show that this condition is guaranteed to exist for any LP solution with integer infeasible flow.

Claim 2. Consider any LP solution of a multi-commodity flow problem with integer infeasible flow of commodity k ∈ K and any st-cut of commodity k, with a set of cut arcs δ(S) and δ(\bar{S}). If there exists at least one arc a ∈ δ(S) such that 0 < f^k_a < 1, then there exists a partition of δ(S) and δ(\bar{S}) into the sets δ_1(S), δ_2(S), and δ_1(\bar{S}), δ_2(\bar{S}) such that

0 < F_1 - B_1 < 1.

Proof. A valid partition of δ(S) and δ(\bar{S}) would be: δ_1(S) = \{a\}, δ_2(S) = δ(S) \setminus \{a\} and δ_1(\bar{S}) = \emptyset, δ_2(\bar{S}) = δ(\bar{S}). By construction, F_1 - B_1 = f^k_a is between zero and one. Thus the claim holds. □

This means that for any non-integer solution to the relaxation of an ODIMCFP, there exists a partition of st-cut arcs, δ_1(S) and δ_1(\bar{S}), such that F_1 - B_1 ≥ 1 and
$F_1 - B_1 \leq 0$ are both violated. Thus, by Theorem 1 we can make a branching rule $F_1 - B_1 \leq 0$ and $F_1 - B_1 \geq 1$ that partitions the feasible set.

As we developed branching rules based on Theorem 1 and Claim 2, we realized that we did not have to limit ourselves to choose flow variables based on $st$-cuts and $st$-cut arcs but could incorporate any two sets of flow variables.

**Theorem 2.** Consider any ODIMCFP over a graph $G = (N, A)$, with a set of commodities $K$, and any two sets of flow variables

$$
\delta_1 = \{(i_1, j_1, k_1), (i_2, j_2, k_2), \ldots, (i_h, j_h, k_h) \in A \times K\}
$$

$$
\delta_2 = \{(w_1, y_1, q_1), (w_2, y_2, q_2), \ldots, (w_p, y_p, q_p) \in A \times K\}.
$$

Then for any feasible solution of the problem set and any non-integer number $r$, exactly one of the following two situations is true:

$$
\sum_{(i,j,k) \in \delta_1} f^k_{ij} - \sum_{(i,j,k) \in \delta_2} f^k_{ij} \leq \lfloor r \rfloor
$$

$$
\sum_{(i,j,k) \in \delta_1} f^k_{ij} - \sum_{(i,j,k) \in \delta_2} f^k_{ij} \geq \lceil r \rceil.
$$

**Proof.** Clearly the sum of positive and negative 0-1 variables can only have one value and must be integer, thus one and only one of the two cases must be true. □

**Claim 3.** For any LP solution of an ODIMCFP, if the solution is not integer feasible then there exists a set of flow variables $\delta_1$ and $\delta_2$ such that:

$$
\sum_{(i,j,k) \in \delta_1} f^k_{ij} - \sum_{(i,j,k) \in \delta_2} f^k_{ij} = r \notin \mathbb{Z}.
$$
Proof. If the LP solution is non-integer then there exists at least one flow variable $f^k_a$, $a \in A$, $b \in K$, with fractional flow. Let $\delta_1$ contain only that flow variable and let $\delta_2$ be the empty set. This implies that $\sum_{(i,j,k) \in \delta_1} f^k_{ij} - \sum_{(i,j,k) \in \delta_2} f^k_{ij} = f^k_a \not\in \mathbb{Z}$. □

Claim 3 guarantees that when solving the LP relaxation of a branch and bound node, if the solution is non-integer then there must exist two sets of flow variables $\delta_1$ and $\delta_2$ such that both constraints in Theorem 2 are violated. We could then eliminate infeasible solutions by using the branching rule

$$
\sum_{(i,j,k) \in \delta_1} f^k_{ij} - \sum_{(i,j,k) \in \delta_2} f^k_{ij} \geq \lceil r \rceil \quad \text{and} \quad \sum_{(i,j,k) \in \delta_1} f^k_{ij} - \sum_{(i,j,k) \in \delta_2} f^k_{ij} \leq \lfloor r \rfloor.
$$

This is important because it allows us to expand our branch and bound rules to incorporate flow variables across multiple commodities.

We often considered the following properties during the development of our branching rules.

Claim 4. For any given sets of flow variables $\delta_1$ and $\delta_2$ containing 0-1 flow variables

$$
-|\delta_2| \leq \sum_{(i,j,k) \in \delta_1} f^k_{ij} - \sum_{(i,j,k) \in \delta_2} f^k_{ij} \leq |\delta_1|.
$$

Claim 5. For any feasible solution of an ODIMCFP over a graph $G = (N, A)$ and a set of commodities $K$,

$$
\sum_{j: (i,j) \in A} f^k_{ij} \leq 1 \quad \forall i \in N, \forall k \in K.
$$

Theorem 3. For any solution $(\bar{f}, \alpha^*)$ of an LP relaxation for the minimum congestion ODIMCFP, if the LP is optimal then there exists at least one arc with congestion ratio equal to $\alpha^*$ with no cyclic flow.

Proof. Assume that the maximum congestion of any arc in the network is $\beta$ and $\beta < \alpha^*$. Clearly the objective function could be decreased to $\beta$ and remain feasible. Thus for a solution to be optimal, at least one arc has congestion $\alpha^*$. Consider that all
arcs with congestion $\alpha^*$ contain cyclic flow. The cycle of flow is not bounded by any of the constraints and could thus be removed from the model. With the cyclic flow removed then there would not exist an arc with congestion $\alpha^*$ and thus the solution could not be optimal.

5.2. Considerations for Developing a Branching Rule

For a problem $P$ and an infeasible solution $x^*$, a branching rule consists of two inequalities $a^T x \geq b$ and $c^T x \geq d$ that are both violated by $x^*$, i.e., $a^T x^* < b$ and $c^T x^* < d$. Problem $P$ is partitioned into $P_1 = \{x : a^T x \geq b, x \in P\}$ and $P_2 = \{x : c^T x \geq d, x \in P\}$. In general, the key for a good branching rule is to make the feasible region of the linear programming relaxation as close as possible to the feasible set of integer solutions [9]. For each of our branching rules, we experimented with promoting various conditions that we believed would make a good branching rule. These conditions were:

- an even partition of the feasible set;
- making it easier to find an integer solution;
- limiting symmetry and cyclic flow.

Often times there is a trade off between developing a branching rule that forces integer convergence and a balanced branch and bound tree. Figures 5.2.1 and 5.2.2 give a visual interpretation of the difference between a balanced and an unbalanced branch and bound tree. For both figures, consider the top box as the feasible region of the relaxation, and each sub-box as the relaxed feasible region of each branch. The unbalanced tree would have very tight bounds on the right side but very loose bounds on the left side. As a result, the right side of the branch and bound tree would likely promote integer convergence much more efficiently than the left side. In contrast, the balanced branch and bound tree provides an even distribution of the feasible set at
each branch. After three branches, the balanced tree has eight relatively small sets of solutions while the unbalanced one ranges from very large (left) to very small (right).

![Figure 5.2.1. A balanced branch and bound tree](image1)

![Figure 5.2.2. An unbalanced branch and bound tree](image2)

It is often better to have a more balanced branch and bound tree. A branch and bound algorithm terminates if the lower bound equals the upper bound. The lower bound is the minimum objective value among all open branch and bound nodes of the LP relaxation and the upper bound is the best integer feasible solution of the branch and bound tree. Having tighter bounds on a relaxed solution will raise the lower bound. Consider Figure 5.2.2, the large box in the bottom left would likely have fewer binding restrictions than any of the other bottom branches of either tree. As a result, the solution of the large box is much more likely to be non-integer and
have a smaller objective function (closer to that of the original relaxation) than any of the other bottom branches. Thus, the balanced branch and bound tree would likely have a better lower bound than the unbalanced branch and bound tree. Although it is possible for the upper bound of the unbalanced branch and bound tree to be optimal, it may not be proved optimal for a long time because the lower bound needs to increase.

Cyclic flow and symmetry was also something that we took into account when creating branching rules. As discussed earlier, cyclic flow can result in redundant branching and lead to problem symmetry. It is often possible that cyclic flow would satisfy a branching constraint rather than the actual flow from the demands. As a result, the branch would have very little change. We experimented with various methods to try and mitigate the effects of redundancy while branching.
CHAPTER 6

Branching Techniques

6.1. Terminology

In the description of each of our branching rules, we use various terms to describe how the rules were implemented. In order to avoid repeating their definition, we present them here.

We refer to a commodity as having non-feasible flow if the commodity does not have an acyclic source to sink path such that every flow variable along that path is one. As stated in Section 3.4, it is possible for a commodity to have an integer feasible solution and also have non-integer flow variables because of cyclic flow. For this reason, we distinguish between a commodity with non-feasible flow and a commodity with an integer feasible path but integer infeasible cyclic flow.

During branch and bound, we often select flow variables for branching based on the optimal solution of the LP relaxation. For all arcs \((i, j) \in A\) and commodities \(k \in K\), we refer to the flow variable as \(f_{ij}^k\) and the value of the variable for the LP relaxation as \(\bar{f}_{ij}^k\).

We refer to an arc as saturated if its congestion equals the objective value and as unsaturated if the congestion is less than the objective. In other words, given a solution \(\bar{f}\) with objective value \(\alpha\), an arc \((i, j) \in A\) as saturated if \(\sum_{k \in K} v_k \bar{f}_{ij}^k = \alpha u_{ij}\) and unsaturated if \(\sum_{k \in K} v_k \bar{f}_{ij}^k < \alpha u_{ij}\).

We refer to a path of a commodity \(k \in K\) with non-feasible flow as the most promising path if it forms an acyclic origin-destination path with positive flow and sends more flow than any other path of that commodity. Intuitively, the path sending
the most flow will likely be the optimal path, hence it is the most promising path. We identified the most promising path of a commodity \(k\) by finding the set \(P\) of all acyclic origin-destination paths with positive flow of commodity \(k\), i.e., \(f^k_{ij} > 0\) for all \((i, j) \in p\), for all \(p \in P\). The most promising path was then identified as the path \(p^* \in P\) with the largest minimal flow variable, i.e., for all \(p \in P \setminus p^*\) the exists an \((i, j) \in p\) such that \(f^k_{ij} \leq f^k_{l'h}\) for all \((l, h) \in p^*\). If there was a tie for most promising path, we gave preference to the shorter path (fewer arcs in path). We often assigned flow variables for branching based on the most promising path.

6.2. Single Variable Rule

The single variable branching rule is the first method from the literature [3] that we considered. It uses a standard branch and bound approach outlined in Section 2.4. The rule identifies a flow variable with fractional flow and creates two branches: a branch with the flow variable equal to one and a branch with the flow variable equal to zero. For instance, if flow variable \(f^k_{ij}, (i, j) \in A, k \in K\), was identified for branching (because it was non-integer), then the added constraint for the two branches would be:

\[
\begin{align*}
&f^k_{ij} = 0 \quad \text{or} \quad f^k_{ij} = 1.
\end{align*}
\]

For this branching rule, the two subproblems maintain all integer solutions of the solution set because the final solution will either have \(f^k_{ij} = 1\) or \(f^k_{ij} = 0\). This rule also provides a partition of the solution set because it is impossible for both branches to share an integer feasible solution, i.e., it is impossible for a feasible solution to have \(f^k_{ij} = 1\) and \(f^k_{ij} = 0\).

A downside of this branching rule is that it can create a very unbalanced branch and bound tree. For a relatively dense graph, the proportion of the paths in the feasible set that do not pass through arc \((i, j)\) would be much larger than the proportion
of feasible paths that pass through that arc. By restricting one side of the branch and bound tree to have flow through a single arc, the number of integer feasible solutions of that side of the branch and bound tree would be substantially less than the side that does not use that arc. This would result in an unbalanced branch and bound tree.

For the implementation of this rule, we chose to select the flow variable from the largest commodity with non-feasible flow. The idea is that a change in flow of a larger commodity would have greater impact on the solution than a smaller commodity. From that commodity, we found the most promising path, i.e., the path that sends the most flow. From that path, we selected the flow variable with the least amount of flow. We selected this flow variable to try and promote integer convergence along the most promising path.

6.3. Node Connection Cover Rule

The node connection cover branching rule is our second rule from the literature, introduced by Barnhart et al. [3]. Consider an ODIMCFP over a graph $G = (N, A)$ and a set of commodities $K$. The idea of the rule comes from the fact that for a single commodity, the total flow that passes through a node is binary, i.e., for a node $d \in N$ and a commodity $k \in K$, $\sum_{j: (d,j) \in A} f^k_{dj} \in \{0, 1\}$.

The branching rule begins by identifying a node $d \in N$, called the divergent node, such that there exists more than one flow variable of a single commodity $k \in K$ leaving node $d$ with positive flow, i.e., $\bar{f}^k_{dn_1} > 0$ and $\bar{f}^k_{dn_2} > 0$ for $n_1 \neq n_2$ and $(d, n_1), (d, n_2) \in A$. Let $S$ denote the set of all arcs whose tail node is $d$, and let $a_1 = (d, n_1)$ and $a_2 = (d, n_2)$. Arcs $a_1$ and $a_2$ are referred to as the divergent arcs. The set $S$ is then partitioned into two subsets with roughly equal cardinality, $U$ and $S \setminus U$, with the condition that $a_1 \in U$ and $a_2 \in S \setminus U$. From these two sets the branching rule can be formed as:
\[ \sum_{(i,j) \in S \setminus U} f_{ij}^k = 0 \quad \text{or} \quad \sum_{(i,j) \in U} f_{ij}^k = 0. \]

The branching rule preserves all feasible solutions of the set because at most one flow variable corresponding to the set would equal one. If the best integer solution has \( f_{m'}^k = 1 \), for \( m' \in U \), then the first branch contains the integer solution, if \( f_{m''}^k = 1 \), \( m'' \in S \setminus U \), then the second branch contains the solution. This rule is valid because it is impossible for \( f_{m'}^k = 1 \) for a \( m' \in U \) and \( f_{m''}^k = 1 \) for some \( m'' \in S \setminus U \). This branching rule creates a much more balanced branch and bound tree, in that both branches retain a large number of feasible solutions.

The down side of this rule is that it creates two subproblems with several (possibly optimal) solutions in common, and thus does not partition the feasible set. Indeed, it is possible for a feasible solution of the problem to exist on both sides of the branches, i.e., there could exist solutions with \( f_{h}^k = 0 \) for all \( h \in S \). If the optimal solution has \( f_{h}^k = 0 \) for all \( h \in S \) then this branch would be non-binding, i.e., redundant. This could potentially double the size of the branch and bound tree.

We implemented the branching rule in the following way. First, we find the commodity \( k \in K \) with non-feasible flow with maximum \( v_k \). We then identify the most promising path \( p_1 \) and the second most promising path \( p_2 \) of commodity \( k \). Next, we select the divergent arcs \( a_1 \) and \( a_2 \) to be the first arcs that are not shared by \( p_1 \) and \( p_2 \). The divergent node \( d \in N \) would thus be the tail node of both \( a_1 \) and \( a_2 \). Letting \( S \) denote the set of all arcs leaving node \( d \), we randomly partition \( S \) into two roughly equal sets \( U \) and \( S \setminus U \) such that \( a_1 \in U \) and \( a_2 \in S \setminus U \). From these two sets, we implement the branching rule outlined above. By implementing the branching rule in this way, we hoped to increase the likelihood that an optimal solution would travel through the divergent node, thus decreasing the chance of non-binding constraint.
6.4. Balanced *st*-cut Rule

The first custom branching rule was designed to create a balanced branch and bound tree based on Theorem 1. It finds a commodity $k \in K$ with non-feasible flow, makes an *st*-cut of the nodes by partitioning $N$ into two sets $S$ and $\bar{S}$, and then partitions the set of cut arcs $\delta(S)$ and $\delta(\bar{S})$ into the sets $\delta_1(S)$, $\delta_2(S)$ and $\delta_1(\bar{S})$, $\delta_2(\bar{S})$ so that $0 < \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij}^k - \sum_{(i,j) \in \delta_1(\bar{S})} \bar{f}_{ij}^k < 1$. The branching rule is then formed in the following way:

\[
\sum_{(i,j) \in \delta_1(S)} f_{ij}^k - \sum_{(i,j) \in \delta_1(S)} f_{ij}^k \geq 1
\]
\[
\sum_{(i,j) \in \delta_1(S)} f_{ij}^k - \sum_{(i,j) \in \delta_1(\bar{S})} f_{ij}^k \leq 0.
\]

We designed our branching rule to promote balance and mitigate the effects of redundancy in the following way:

- To create an *st*-cut such that the most promising paths of a commodity would have a single arc in the set of cut arcs and for that arc to be infeasible.
- To partition the cut arcs so that approximately half of the paths with flow would be in the feasible set of each branch.
- To create an *st*-cut that provides a large number of cut arcs.
- To partition the cut arcs such that for every $(i,j) \in \delta_1(S)$ the reverse arc would be assigned $(j,i) \in \delta_1(\bar{S})$.
- To have $|\delta_1(S)| \approx |\delta_2(S)|$ and similarly $|\delta_1(\bar{S})| \approx |\delta_2(\bar{S})|$.

We decided to create the *st*-cut such that the most promising paths would have a single infeasible arc in the set of cut arcs to limit redundancy and promote change. Consider a most promising path $p$ having 3 arcs across the set of cut arcs (2 forward and 1 backward), and the corresponding flow variables having infeasible flow. If the
two forward arcs were in the set $\delta_1(S)$ and the backward arc was in the set $\delta_2(S)$, then any change in flow of the path $p$ would be added twice in the branching constraint. If $p$ ended up being the optimal path, the constraint would be non-binding. Thus the branch would be redundant.

For every arc $(i, j) \in \delta_1(S)$ we assigned the reverse arc $(i, j) \in \delta_1(\bar{S})$. The idea was to try to stop the ability for cyclic flow to satisfy the branching constraints. Consider a single unsaturated arc $(h, w) \in \delta_1(S)$ (i.e., its congestion is less than the objective value) and suppose the reverse arc $(w, h)$ is in $\delta_2(\bar{S})$ and is also unsaturated. The branching constraint $\sum_{(i,j) \in \delta_1(S)} f^k_{ij} - \sum_{(i,j) \in \delta_1(\bar{S})} f^k_{ij} \geq 1$ could be satisfied by creating the cycle of flow $f^k_{h,w} = 1$ and $f^k_{w,h} = 1$. As a result, there would be no change in the flow from the flow conservation constraints and thus no real change in flow in the problem. By setting the reverse arc of every arc in $\delta_1(S)$ to the set $\delta_1(\bar{S})$ the branching constraint could not be satisfied from a cycle of flow.

We chose to make a cut with a large number of cut-arcs and to make the partitions of cut-arcs approximately equal to promote a balanced branching rule. Intuitively, the feasible set of paths would be more spread out among a larger cut, thus an even partition of the arcs would have a greater likelihood of dividing the feasible set evenly.

We implemented this branching rule in the following way. Let $k \in K$ be the largest commodity with non-feasible flow. Let $P$ be the set of origin-destination paths of commodity $k$ with positive flow such that $p_1 \in P$ is the most promising path, $p_2 \in P$ is the second most promising, i.e., the path that sends the second largest amount of flow of commodity $k$, and analogously, $p_i \in P$ is the $i$th most promising. For each path $p_i \in P$ let $p_i = n^i_1 - n^i_2 - \cdots - n^i_{m_i}$, $n^i_j \in N$ for all $j = 1, 2, \ldots, m_i$ (note that this implies that $n^i_1 = s_k$ and $n^i_{m_i} = t_k$ for all $p_i \in P$) and let $(n^i_y, n^i_{y+1}) \in A$ be the first arc along the path of $p_i$ such that the corresponding flow variable of commodity $k$ is non-integer, i.e., $\bar{f}^{k}_{n^i_j n^i_{j+1}} = 1$ for all $j = 1, 2, \ldots, y-1$ and $\bar{f}^{k}_{n^i_y n^i_{y+1}} < 1$. 

37
The $st$-cut sets $S$ and $\bar{S}$ were initialized with $n^i_1 - n^i_2 - \cdots - n^i_y \in S$ and $n^i_{y+1} - n^i_{y+2} - \cdots - n^i_{m_i} \in \bar{S}$ for all $i = 1, 2, \ldots, |P|$. It is possible that cyclic flow, along the union of acyclic paths, results in a node in both $S$ and $\bar{S}$. If this happens, we gave higher priority to the path that sends more flow, i.e., the path that is more promising. Figure 6.4.1 depicts three origin-destination paths $s - 1 - 2 - t$, $s - 2 - t$ and $s - t$ with minimum flow .9, .3, and .1 respectively. In this example, the most promising path is $s - 1 - 2 - t$, which has integer feasible flow $s - 1 - 2$. For our rule, the $st$-cut would be initialized with $S = \{s, 1, 2\}$ and $\bar{S} = \{t\}$.

![Figure 6.4.1. Example of cyclic flow forming an origin-destination path](image)

With this partially formed $st$-cut, we would begin to partition the set of $st$-cut arcs $\delta(S)$ into the sets $\delta_1(S)$ and $\delta_2(S)$. Note that for this rule, for every $(i, j) \in \delta_1(S)$ we have the reverse arc $(j, i) \in \delta_1(\bar{S})$; this is assumed throughout the description. The set of cut-arcs $\delta(S)$ is partially partitioned in the following way:

$$(n^i_y, n^i_{y+1}) \in \delta_1(S), \forall p_i \in P : n^i_y \in S, n^i_{y+1} \in \bar{S}, i \text{ odd};$$

$$(n^i_y, n^i_{y+1}) \in \delta_2(S), \forall p_i \in P : n^i_y \in S, n^i_{y+1} \in \bar{S}, i \text{ even}.  \quad \text{(38)}$$

The rest of the unassigned cut nodes were added to the $st$-cut in the following way. If $|S| < |\bar{S}|$ then find an unassigned node with at least one arc connection to $S$ and add the node to $S$. If no such node exists, randomly assign an unassigned node to $S$. If $|S| \geq |\bar{S}|$, find an unassigned node with at least one arc connection to $S$ and
add the node to $\bar{S}$. If no such node exists, randomly assign an unassigned node to $\bar{S}$. This method of assigning nodes to the $st$-cut provides a heuristic approach for making an $st$-cut with a large number of cut arcs.

The rest of the unassigned cut arcs were then partitioned randomly into the sets $\delta_1(S)$ and $\delta_2(S)$ in the following way. If $|\delta_1(S)| \geq |\delta_2(S)|$ find an unassigned cut arc $(h, w)$ and add it to $\delta_2(S)$ (thus arc $(w, h)$ goes to $\delta_2(\bar{S})$). If $|\delta_1(S)| < |\delta_2(S)|$ then an arc would be added to $\delta_1(S)$, but there was more consideration with the selection.

If $|\delta_1(S)| < |\delta_2(S)|$ and $0 \leq \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij} - \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij} < 1$, choose any unassigned cut arc $(h, w) \in \delta(S)$ and add it to $\delta_1(S)$.

If $|\delta_1(S)| < |\delta_2(S)|$ and $\sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij} - \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij} < 0$, search for an unassigned cut arc $(h, w) \in \delta(S)$ such that $\bar{f}_{hw} - \bar{f}_{wh} > 0$ and add it to $\delta_1(S)$. If no arc exists, then search the set $\delta_2(S)$, starting with the last arc added, for an arc $(h, w) \in \delta_2(S)$ satisfying $\bar{f}_{hw} - \bar{f}_{wh} > 0$ and add it to $\delta_1(S)$. If again no arc exists, then find an arc $(h, w) \in \delta_1(S)$ such that $\bar{f}_{hw} - \bar{f}_{wh} < 0$ and add it to $\delta_2(S)$.

If $|\delta_1(S)| < |\delta_2(S)|$ and $1 < \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij} - \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij}$ the algorithm would do an entirely similar process of searching for an arc $(h, w)$ and reverse arc $(w, h)$ such that $\bar{f}_{hw} - \bar{f}_{wh} < 0$ and adding it to the set $\delta_1(S)$.

After all cut arcs were partitioned into the sets $\delta_1(S)$ and $\delta_2(S)$, if $0 < \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij} - \sum_{(i,j) \in \delta_1(S)} \bar{f}_{ij} < 1$ was not satisfied, the algorithm would search $\delta_2(S)$ for an arc that would satisfy the constraint and move it to $\delta_1(S)$. If none existed, the algorithm would search $\delta_1(S)$ for an arc whose removal would satisfy the constraint and move it to $\delta_2(S)$. With the cut arcs partitioned in this way, the constraint would be satisfied and we could implement the branching rule as defined above.

### 6.5. Shortest Path Rule

The second method we tried was again based on Theorem 1, i.e., it makes a partition of $st$-cut arcs. The method was designed to eliminate infeasible shortest
paths from a commodity with infeasible flow. This rule begins by finding the shortest acyclic path $p$ of positive flow variables for the commodity $k \in K$ with maximal $v_k$ and non-feasible flow. Here, the shortest path is referring to the number of arcs the commodity must travel through to go from source to sink node. Let $n_1 - n_2 - \cdots - n_p$ represent the sequence of nodes for this shortest path $P$ and let $(n_w, n_{w+1})$, $w < p$, be the first arc such that $\bar{f}_{n_w,n_{w+1}}^k < 1$. An s-t cut is created such that $\bar{S} = \{n_{w+1}, n_{w+2}, \ldots, n_p\}$ and $S = V \setminus \bar{S}$. The rule then constructs the partition of cut-arcs $\delta_1(S)$ and $\delta_1(\bar{S})$ as follows:

$$\delta_1(S) = \{(i, j) : i \in S \cap P, j \in \bar{S}\} \quad \text{and} \quad \delta_1(\bar{S}) = \emptyset.$$ 

Since $P$ is the shortest path with positive flow, $\bar{f}_{n_w,n_{w+1}}^k$ is the only flow variable with positive flow in the set $\{\bar{f}_{i,j}^k : (i, j) \in \delta_1(S)\}$, otherwise there would exist a shorter path with positive flow from source to sink. By partitioning the arcs in this way, the current LP relaxation would have $F_1 := \sum_{(i,j) \in \delta_1(S)} f_{i,j}^k = \bar{f}_{n_w,n_{w+1}}^k$ and $B_1 := \sum_{(i,j) \in \delta_1(\bar{S})} f_{i,j}^k = 0$. Thus $0 < F_1 - B_1 < 1$ and the branching rule $F_1 - B_1 \geq 1$ or $F_1 - B_1 \leq 0$ can be applied.

Theoretically, this method creates a branch and bound tree that is at least as balanced as the single variable branching rule. If $\delta_1(S)$ contains only one arc then the branching rule would be equivalent to setting $f_{n_w,n_{w+1}}^k = 1$ and $f_{n_w,n_{w+1}}^k = 0$, and would thus be equivalent to the single variable method. If $\delta_1(S)$ contains more than one arc, the branching rule would be forcing a set of variables to zero on one branch and forcing at least one variable to have positive flow on the other. By branching on a set of variables, the two branches would retain a more balanced distribution of integer feasible solutions.

This branching strategy attempts to force integer convergence along the side $F_1 - B_1 \geq 1$ while remaining more balanced than the single variable method. By selecting
the branches in this way, we attempt to mitigate the effects of cyclic flow by forcing the increase of flow to stay on the path. The downside of this is that we may not be selecting flow variables that are as good for branching as the other methods.

### 6.6. Single Path Rule

The single path branching rule was designed to promote integer convergence along a single path of a commodity by forcing flow on that path. This is our most unbalanced branching method which could affect the amount of time to find an optimal solution in the long run (as discussed in the branching strategies). A possible benefit of this method is that it increases the chance for an immediate feasible solution on one side of the branch and bound tree and thus increases the chance of a new upper bound.

The rule begins by finding the commodity \( k \in K \) with maximal \( v_k \) and non-feasible flow. From commodity \( k \), we identify the most promising path \( p = a_1 - a_2 - \cdots - a_h, \) \( a_i \in A, \) for all \( i = 1, \ldots, h. \) The rule then adds all arcs \( a_1, a_2, \ldots, a_w, w \leq h, \) to \( \delta_1 \) such that the rounded up sum of flow variables in \( \delta_1 \) equals the number of arcs in \( \delta_1, \) i.e., \( a_w \in \delta_1 \) if \( [\sum_{i=1}^{w} f_{a_i}^k] = w, w \leq h. \) By assigning arcs in this way, we guarantee that \( \sum_{(i,j) \in \delta_1} f_{ij}^k \notin \mathbb{Z} \) and \( |\delta_1| = [\sum_{(i,j) \in \delta_1} f_{ij}^k]. \) Thus by Theorem 2 we can make the following branches:

\[
\sum_{(i,j) \in \delta_1} f_{ij}^k = |\delta_1|
\]

\[
\sum_{(i,j) \in \delta_1} f_{ij}^k \leq |\delta_1| - 1.
\]

To demonstrate the unbalancedness of the rule, the feasible set of paths for commodity \( k \) of branch \( \sum_{(i,j) \in \delta_1} f_{ij}^k = |\delta_1| \) could potentially be of cardinality one (the path \( p \)), in which case the set of feasible paths for the branch \( \sum_{(i,j) \in \delta_1} f_{ij}^k \leq |\delta_1| - 1 \) would
have all other paths, and thus an exponentially large number of paths. This could greatly increase the number of subproblems needed for a provably optimal solution.

One of the key benefits of this rule is its application toward branch and price. As we will discuss in Chapter 8, this rule can be added to the constraints of a pricing problem and remain tractable during the column generation phase.

6.7. Congested Arc Rule

Based on our experiments, the congested arc rule gave us the best results of any of our other branching rules. The general idea of the congested arc branching rule consists of finding the most congested arc in the graph and making two branches, one branch forcing more commodities through the arc and the other branch forcing less. As we will show, this idea creates a rule that greatly increases the chance of raising the lower bound.

From Theorem 3 we know that for the LP relaxation to be optimal there must be at least one arc that is fully saturated with no cyclic flow. Any change in flow of this arc is very likely to increase the value of the objective function. Suppose that there is a small increase in flow through the most congested arc. Clearly the objective function would have to increase to satisfy the increased size of the capacity constraints. Suppose instead that there is a small decrease in the amount of flow through the arc. Then there is a high likelihood that the objective value would increase as well. The reasoning is simple: if flow could easily be diverted to another arc without raising the objective, then there would be no reason for that arc to be fully saturated to begin with. In fact, if the flow decreases through the most congested arc and the objective does not change, then there must exist another arc that is fully saturated at the current objective value and is not influenced by the change in flow of the chosen arc. If this was not true, a small decrease in flow through the most congested arc
would result in a decrease in the objective function, thus contradicting that the LP solution is optimal.

Ideally, for our branching rule, the total flow through the most congested arc would always contain infeasible flows and the total flow through that arc would always be non-integer. This is not always the case, thus we had to design our rule to account for these nuances, resulting in a rule that would sometimes select the second or third most congested arc of the LP solution.

The rule works by finding the most congested arc with at least one infeasible flow variable. The congestion of an arc is calculated by finding the total amount of acyclic flow through the arc and dividing the acyclic flow by the capacity. Accounting for the cyclic flow when calculating the most congested arc decreases the likelihood that the branch could be satisfied by diverting cyclic flow away from the arc, thus limiting the possibility of redundant branching.

The total amount of acyclic flow through an arc \((i, j) \in A\) was calculated by summing the difference in flow of arc \((i, j)\) and reverse arc \((j, i)\) for every commodity \(k \in K\) such that the difference was positive. The total acyclic flow through the arc \((i, j)\) could be calculated:

\[
\omega_{ij} := \sum_{k \in K} v_k \max\{\bar{f}_{ij}^k - \bar{f}_{ji}^k, 0\}.
\]

The congestion of the arc could then be calculated by dividing the total acyclic flow by the capacity of that arc:

\[
\omega_{ij} = \frac{\omega_{ij}}{u_{ij}}.
\]

After scanning all arcs, we select the arc \((i, j) \in A\) that has the most congestion and at least one infeasible flow variable. Note that this arc may or may not be the most congested arc of the LP solution because of the infeasible flow requirement. If
the sum of flow variables was non-integer we would branch, i.e., if \( \sum_{k \in K} \bar{f}^k_{ij} = r \notin \mathbb{Z}^+ \) we would create the branches:

\[
\sum_{k \in K} f^k_{ij} \leq \lfloor r \rfloor \\
\sum_{k \in K} f^k_{ij} \geq \lceil r \rceil.
\]

If \( \sum_{k \in K} \bar{f}^k_{ij} = r \in \mathbb{Z}^+ \), then we would find the flow variable closest to .5 and remove it. This flow variable must exist because there must exist at least one variable with infeasible flow. This was done by calculating \( \max_{k \in K} \bar{f}^k_{ij} \cdot (1 - \bar{f}^k_{ij}) \). The idea is that removing the flow variable closest to .5 would create the most change in flow among the two branches. Letting \( q \in K \) denote the commodity of the flow variable closest to .5, the sum of all flows through the arc \((i,j)\) minus the flow of commodity \( q \) through arc \((i,j)\) would be non-integer, i.e., \( \sum_{k \in K \setminus q} \bar{f}^k_{ij} = w \notin \mathbb{Z}^+ \). We would create the branches:

\[
\sum_{k \in K \setminus q} f^k_{ij} \leq \lfloor w \rfloor \\
\sum_{k \in K \setminus q} f^k_{ij} \geq \lceil w \rceil.
\]
CHAPTER 7

Experimental Results

7.1. Computation

All tests were conducted using IBM ILOG CPLEX Optimization Studio V12.3 64 bit. Each instance was run on a single node and executed on a single thread of Clemson University’s Palmetto cluster. The cluster consists of both multi-core shared systems and multi-core distributed memory architecture. Each node has a processor speed between 2.1-2.6 GHz and between 12-48 GB of RAM. The operating system is Scientific 6 Linux. The program was written in C++ and compiled using g++ with compile option -O3. Although CPLEX 12.3 has the capability of running with multiple processors, all of our branch and bound procedures only used one because of the implementation of nontrivial callback functions in our code.

To create an ODIMCFP in CPLEX, we designed a program to read a specially formatted data file which contained all the network information of the problem and store that information into a user defined class in C++. From this class, we created multiple functions to access specific information about the network. With this information, we were able to build the model in CPLEX.

For the implementation of these branching rules in CPLEX, we used a combination of the duplicate callback function with the branch callback class. The branch callback class allows the user to implement user written branching rules during the branch and bound process. The duplicate callback allows the user to retrieve and store information while in a callback class. We used the duplicate callback to store all the information pertaining to model and to keep track of the maximum node depth.

45
We used a heuristic callback coupled with the duplicate callback function to check for feasibility at each branch. As stated in Section 3.4, it was often possible for CPLEX to interpret an integer feasible solution as infeasible because of non-integer cyclic flow. The information of the model was stored in the duplicate callback. This information allows CPLEX to check if the LP solution has an integer feasible solution that satisfies the flow conservation constraints from within the heuristic callback. If the LP solution had an integer feasible solution, then the integer solution was injected into the solution pool using the set solution function of the heuristic callback class. If the LP solution did not have a feasible solution, then the heuristic would do nothing and the program would proceed to the branch callback procedure.

Each of the branching rules used the same program to build the model and implement the heuristic. This consisted of a few hundred lines of code. The code for building the branching rule (in the branch callback) varied based on the complexity of the rule. The most complex branching rule was the balanced st-cut, which consisted of a few hundred lines. The least complex rule consisted of about a hundred lines. All code is available upon request.

All default parameters were left on during implementation. Some of these parameters include: model pre-solve, linear reductions, node pre-solve, node dives, as well as periodic application of various cut separators and heuristics. For further information on the default CPLEX parameters, and the control callback classes, we refer the reader to the CPLEX user manual [6].

7.2. Targeted Networks

Optimization methods often find their toughest test bed on real-word problems; we have chosen to apply our algorithms to both fictitious and real topologies and origin-destination pairs. For information pertaining to the origin of these topologies, we refer the reader to Belotti and Pinar [4]. These topologies did not include the
capacity of each arc or the demand of each commodity. For each of the topologies, each origin-destination pair was randomly assigned a supply/demand between 10 and 16, and each arc capacity was randomly assigned a value between 10 and 30. Figure 7.2.1 depicts the number of nodes, arcs, and commodities of each of the network topologies used.

![Table of Network Topologies](image)

**Figure 7.2.1.** Topology of problem instances

We examined each of our branching rules, as well as the CPLEX default MIP solver against the topologies. For each instance, CPLEX ran until the solution converged to an optimal solution or until a CPU time of 1.5 hours elapsed. The next three pages display the results of our branching rules. For each rule, we record the number branch and bound nodes, the maximum depth, the lower bound (the objective value of the worst infeasible solution of active nodes), and the upper bound (best integer feasible solution) of the branch and bound tree. In the last column of a branching rule, if the
instance converged to an optimal solution within 1.5 hours of CPU time, we record the time of convergence, otherwise we record the feasibility gap. The feasibility gap is the percent difference between the upper and lower bound of the branch and bound tree. The feasibility gap is differentiated from the time of convergence with a % symbol.
<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower</th>
<th>Upper</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>usld</td>
<td>90000</td>
<td>12.9091</td>
<td>12.9231</td>
<td>0.11%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>130000</td>
<td>14.0091</td>
<td>13.0556</td>
<td>1.12%</td>
</tr>
</tbody>
</table>

Results

Single Variable Shortest Path st-Cut Node Connection Cover

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower</th>
<th>Upper</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>usld</td>
<td>90000</td>
<td>12.9091</td>
<td>12.9231</td>
<td>0.11%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>130000</td>
<td>14.0091</td>
<td>13.0556</td>
<td>1.12%</td>
</tr>
<tr>
<td>Instance</td>
<td>Nodes</td>
<td>Depth</td>
<td>Lower</td>
<td>Upper</td>
<td>Time/Gap</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>Single Path</td>
<td>40000</td>
<td>636</td>
<td>2.2041</td>
<td>0.11%</td>
<td></td>
</tr>
<tr>
<td>Balanced st-Cut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Results

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower</th>
<th>Upper</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>alcatel</td>
<td>60000</td>
<td>313</td>
<td>12.9091</td>
<td>2.175</td>
<td></td>
</tr>
<tr>
<td>arpa-21</td>
<td>56000</td>
<td>311</td>
<td>15.2969</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>arpa-24</td>
<td>56000</td>
<td>311</td>
<td>11.1538</td>
<td>2.59</td>
<td></td>
</tr>
<tr>
<td>bhvd</td>
<td>56000</td>
<td>310</td>
<td>13.0952</td>
<td>2.34</td>
<td></td>
</tr>
<tr>
<td>bhve</td>
<td>56000</td>
<td>310</td>
<td>11.4444</td>
<td>2.72</td>
<td></td>
</tr>
<tr>
<td>latadl</td>
<td>56000</td>
<td>310</td>
<td>15.8276</td>
<td>2.72</td>
<td></td>
</tr>
<tr>
<td>njlata</td>
<td>56000</td>
<td>310</td>
<td>11.1731</td>
<td>2.71</td>
<td></td>
</tr>
<tr>
<td>nsf-56</td>
<td>56000</td>
<td>310</td>
<td>18.4211</td>
<td>2.91</td>
<td></td>
</tr>
<tr>
<td>nsf-90</td>
<td>56000</td>
<td>310</td>
<td>13.6667</td>
<td>2.91</td>
<td></td>
</tr>
<tr>
<td>ny_cep2</td>
<td>56000</td>
<td>310</td>
<td>22.2041</td>
<td>3.17</td>
<td></td>
</tr>
<tr>
<td>san</td>
<td>56000</td>
<td>310</td>
<td>12.03</td>
<td>2.175</td>
<td></td>
</tr>
<tr>
<td>san</td>
<td>56000</td>
<td>310</td>
<td>6.2727</td>
<td>0.384615</td>
<td></td>
</tr>
<tr>
<td>Toronto</td>
<td>56000</td>
<td>310</td>
<td>13.1818</td>
<td>6.3214</td>
<td></td>
</tr>
<tr>
<td>usld</td>
<td>56000</td>
<td>310</td>
<td>15.8571</td>
<td>6.3214</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower</th>
<th>Upper</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Path</td>
<td>40000</td>
<td>636</td>
<td>2.2041</td>
<td>0.11%</td>
<td></td>
</tr>
<tr>
<td>Balanced st-Cut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower</th>
<th>Upper</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>albcatel</td>
<td>60000</td>
<td>313</td>
<td>12.9091</td>
<td>2.175</td>
<td></td>
</tr>
<tr>
<td>alpa-21</td>
<td>56000</td>
<td>311</td>
<td>15.2969</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>alpa-24</td>
<td>56000</td>
<td>311</td>
<td>11.1538</td>
<td>2.59</td>
<td></td>
</tr>
<tr>
<td>bhvd</td>
<td>56000</td>
<td>310</td>
<td>13.0952</td>
<td>2.34</td>
<td></td>
</tr>
<tr>
<td>bhve</td>
<td>56000</td>
<td>310</td>
<td>11.4444</td>
<td>2.72</td>
<td></td>
</tr>
<tr>
<td>latadl</td>
<td>56000</td>
<td>310</td>
<td>15.8276</td>
<td>2.72</td>
<td></td>
</tr>
<tr>
<td>njlata</td>
<td>56000</td>
<td>310</td>
<td>11.1731</td>
<td>2.71</td>
<td></td>
</tr>
<tr>
<td>nsf-56</td>
<td>56000</td>
<td>310</td>
<td>18.4211</td>
<td>2.91</td>
<td></td>
</tr>
<tr>
<td>nsf-90</td>
<td>56000</td>
<td>310</td>
<td>13.6667</td>
<td>2.91</td>
<td></td>
</tr>
<tr>
<td>ny_cep2</td>
<td>56000</td>
<td>310</td>
<td>22.2041</td>
<td>3.17</td>
<td></td>
</tr>
<tr>
<td>san</td>
<td>56000</td>
<td>310</td>
<td>12.03</td>
<td>2.175</td>
<td></td>
</tr>
<tr>
<td>san</td>
<td>56000</td>
<td>310</td>
<td>6.2727</td>
<td>0.384615</td>
<td></td>
</tr>
<tr>
<td>Toronto</td>
<td>56000</td>
<td>310</td>
<td>13.1818</td>
<td>6.3214</td>
<td></td>
</tr>
<tr>
<td>usld</td>
<td>56000</td>
<td>310</td>
<td>15.8571</td>
<td>6.3214</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower</th>
<th>Upper</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Path</td>
<td>40000</td>
<td>636</td>
<td>2.2041</td>
<td>0.11%</td>
<td></td>
</tr>
<tr>
<td>Balanced st-Cut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower</th>
<th>Upper</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>albcatel</td>
<td>60000</td>
<td>313</td>
<td>12.9091</td>
<td>2.175</td>
<td></td>
</tr>
<tr>
<td>alpa-21</td>
<td>56000</td>
<td>311</td>
<td>15.2969</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>alpa-24</td>
<td>56000</td>
<td>311</td>
<td>11.1538</td>
<td>2.59</td>
<td></td>
</tr>
<tr>
<td>bhvd</td>
<td>56000</td>
<td>310</td>
<td>13.0952</td>
<td>2.34</td>
<td></td>
</tr>
<tr>
<td>bhve</td>
<td>56000</td>
<td>310</td>
<td>11.4444</td>
<td>2.72</td>
<td></td>
</tr>
<tr>
<td>latadl</td>
<td>56000</td>
<td>310</td>
<td>15.8276</td>
<td>2.72</td>
<td></td>
</tr>
<tr>
<td>njlata</td>
<td>56000</td>
<td>310</td>
<td>11.1731</td>
<td>2.71</td>
<td></td>
</tr>
<tr>
<td>nsf-56</td>
<td>56000</td>
<td>310</td>
<td>18.4211</td>
<td>2.91</td>
<td></td>
</tr>
<tr>
<td>nsf-90</td>
<td>56000</td>
<td>310</td>
<td>13.6667</td>
<td>2.91</td>
<td></td>
</tr>
<tr>
<td>ny_cep2</td>
<td>56000</td>
<td>310</td>
<td>22.2041</td>
<td>3.17</td>
<td></td>
</tr>
<tr>
<td>san</td>
<td>56000</td>
<td>310</td>
<td>12.03</td>
<td>2.175</td>
<td></td>
</tr>
<tr>
<td>san</td>
<td>56000</td>
<td>310</td>
<td>6.2727</td>
<td>0.384615</td>
<td></td>
</tr>
<tr>
<td>Toronto</td>
<td>56000</td>
<td>310</td>
<td>13.1818</td>
<td>6.3214</td>
<td></td>
</tr>
<tr>
<td>usld</td>
<td>56000</td>
<td>310</td>
<td>15.8571</td>
<td>6.3214</td>
<td></td>
</tr>
<tr>
<td>Instance</td>
<td>Nodes</td>
<td>Depth</td>
<td>Lower Bound</td>
<td>Upper Bound</td>
<td>Time/Gap</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td>--------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>45a</td>
<td>21000</td>
<td>0.11%</td>
<td>12.9091</td>
<td>12.9231</td>
<td>0.11%</td>
</tr>
<tr>
<td>alcatel</td>
<td>22600</td>
<td></td>
<td>15.2969</td>
<td>15.3333</td>
<td>0.30%</td>
</tr>
<tr>
<td>arpa-21</td>
<td>25000</td>
<td></td>
<td>10.1863</td>
<td>10.1863</td>
<td>0.18%</td>
</tr>
<tr>
<td>arpa-24</td>
<td>21000</td>
<td></td>
<td>13.0952</td>
<td>13.0952</td>
<td>0.16%</td>
</tr>
<tr>
<td>bhvd</td>
<td>15000</td>
<td></td>
<td>12.9667</td>
<td>12.9667</td>
<td>0.07%</td>
</tr>
<tr>
<td>bhve</td>
<td>12000</td>
<td></td>
<td>11.4444</td>
<td>11.4444</td>
<td>0.30%</td>
</tr>
<tr>
<td>latadl</td>
<td>6000</td>
<td>0.08%</td>
<td>11.1905</td>
<td>11.1905</td>
<td>0.16%</td>
</tr>
<tr>
<td>njlata</td>
<td>44000</td>
<td>0.26%</td>
<td>18.4286</td>
<td>18.4286</td>
<td>0.19%</td>
</tr>
<tr>
<td>nsf-56</td>
<td>98</td>
<td>0.06%</td>
<td>3.27273</td>
<td>3.27273</td>
<td>0.19%</td>
</tr>
<tr>
<td>nsf-90</td>
<td>12</td>
<td>0.05</td>
<td>3.14286</td>
<td>3.14286</td>
<td>0.12%</td>
</tr>
<tr>
<td>ny_cep2</td>
<td>98</td>
<td>0.19%</td>
<td>3.1154</td>
<td>3.1154</td>
<td>0.19%</td>
</tr>
<tr>
<td>torento</td>
<td>392</td>
<td>0.21%</td>
<td>3.26249</td>
<td>3.26249</td>
<td>0.19%</td>
</tr>
<tr>
<td>usld</td>
<td>398</td>
<td>0.19%</td>
<td>3.07692</td>
<td>3.07692</td>
<td>0.19%</td>
</tr>
</tbody>
</table>

**Congested Arc**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>45a</td>
<td>21000</td>
<td>0.11%</td>
<td>12.9091</td>
<td>12.9231</td>
<td>0.11%</td>
</tr>
<tr>
<td>alcatel</td>
<td>22600</td>
<td></td>
<td>15.2969</td>
<td>15.3333</td>
<td>0.30%</td>
</tr>
<tr>
<td>arpa-21</td>
<td>25000</td>
<td></td>
<td>10.1863</td>
<td>10.1863</td>
<td>0.18%</td>
</tr>
<tr>
<td>arpa-24</td>
<td>21000</td>
<td></td>
<td>13.0952</td>
<td>13.0952</td>
<td>0.16%</td>
</tr>
<tr>
<td>bhvd</td>
<td>15000</td>
<td></td>
<td>12.9667</td>
<td>12.9667</td>
<td>0.07%</td>
</tr>
<tr>
<td>bhve</td>
<td>12000</td>
<td></td>
<td>11.4444</td>
<td>11.4444</td>
<td>0.30%</td>
</tr>
<tr>
<td>latadl</td>
<td>6000</td>
<td>0.08%</td>
<td>11.1905</td>
<td>11.1905</td>
<td>0.16%</td>
</tr>
<tr>
<td>njlata</td>
<td>44000</td>
<td>0.26%</td>
<td>18.4286</td>
<td>18.4286</td>
<td>0.19%</td>
</tr>
<tr>
<td>nsf-56</td>
<td>98</td>
<td>0.06%</td>
<td>3.27273</td>
<td>3.27273</td>
<td>0.19%</td>
</tr>
<tr>
<td>nsf-90</td>
<td>12</td>
<td>0.05</td>
<td>3.14286</td>
<td>3.14286</td>
<td>0.12%</td>
</tr>
<tr>
<td>ny_cep2</td>
<td>98</td>
<td>0.19%</td>
<td>3.1154</td>
<td>3.1154</td>
<td>0.19%</td>
</tr>
<tr>
<td>torento</td>
<td>392</td>
<td>0.21%</td>
<td>3.26249</td>
<td>3.26249</td>
<td>0.19%</td>
</tr>
<tr>
<td>usld</td>
<td>398</td>
<td>0.19%</td>
<td>3.07692</td>
<td>3.07692</td>
<td>0.19%</td>
</tr>
</tbody>
</table>

**Results**

<table>
<thead>
<tr>
<th>Cplex Default Congested Arc</th>
<th>Nodes</th>
<th>Depth</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
<th>Time/Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>45a</td>
<td>21000</td>
<td>0.11%</td>
<td>12.9091</td>
<td>12.9231</td>
<td>0.11%</td>
</tr>
<tr>
<td>alcatel</td>
<td>22600</td>
<td></td>
<td>15.2969</td>
<td>15.3333</td>
<td>0.30%</td>
</tr>
<tr>
<td>arpa-21</td>
<td>25000</td>
<td></td>
<td>10.1863</td>
<td>10.1863</td>
<td>0.18%</td>
</tr>
<tr>
<td>arpa-24</td>
<td>21000</td>
<td></td>
<td>13.0952</td>
<td>13.0952</td>
<td>0.16%</td>
</tr>
<tr>
<td>bhvd</td>
<td>15000</td>
<td></td>
<td>12.9667</td>
<td>12.9667</td>
<td>0.07%</td>
</tr>
<tr>
<td>bhve</td>
<td>12000</td>
<td></td>
<td>11.4444</td>
<td>11.4444</td>
<td>0.30%</td>
</tr>
<tr>
<td>latadl</td>
<td>6000</td>
<td>0.08%</td>
<td>11.1905</td>
<td>11.1905</td>
<td>0.16%</td>
</tr>
<tr>
<td>njlata</td>
<td>44000</td>
<td>0.26%</td>
<td>18.4286</td>
<td>18.4286</td>
<td>0.19%</td>
</tr>
<tr>
<td>nsf-56</td>
<td>98</td>
<td>0.06%</td>
<td>3.27273</td>
<td>3.27273</td>
<td>0.19%</td>
</tr>
<tr>
<td>nsf-90</td>
<td>12</td>
<td>0.05</td>
<td>3.14286</td>
<td>3.14286</td>
<td>0.12%</td>
</tr>
<tr>
<td>ny_cep2</td>
<td>98</td>
<td>0.19%</td>
<td>3.1154</td>
<td>3.1154</td>
<td>0.19%</td>
</tr>
<tr>
<td>torento</td>
<td>392</td>
<td>0.21%</td>
<td>3.26249</td>
<td>3.26249</td>
<td>0.19%</td>
</tr>
<tr>
<td>usld</td>
<td>398</td>
<td>0.19%</td>
<td>3.07692</td>
<td>3.07692</td>
<td>0.19%</td>
</tr>
</tbody>
</table>
7.3. Results with Problem Symmetry

From the results, we saw that the branching rules: single variable, node connection cover, balanced $st$-cut, shortest path $st$-cut, and single path had little to no change on the lower bound. As a result, these rules display traits of “problem symmetry”. Barnhart et al. [3] had similar problems with this symmetry in their branch and price application of our single variable and node connection cover branching rules to an ODIMCFP. This symmetry seems to be inherent for branching rules based on a single commodity of this type of problem. The reason for this symmetry is that commodities that share node pairs and have similar size will simply swap places during the branching process. For example, consider two commodities $k$ and $q$ which have the same supply/demand and share the same node pair. It is often possible that the path of flow for commodity $k$ does not follow the same path of flow as commodity $q$. A branching rule based on a single commodity might restrict $k$ from using its current path. The branching rule could be easily enforced without raising the objective by switching the flow of commodity $k$ with the flow of commodity $q$. As a result, commodity $q$ would have infeasible flow where $k$ used to be and a redundant branch would be implemented [3].

Of the five methods with problem symmetry, the single variable rule had the most success at closing the gap. With the exception of bhv1 and nsf-56, the single variable method did as well as or better than the other branching rules. That being said, most of the time at least one other branching rule would have the same feasibility gap as the single variable rule.

The method that tended to produce the worst results was the balanced $st$-cut rule. This rule had the single worst performance in 13 of the 34 problem instances. This method also tended to produce the fewest number of branches, i.e., the fewest number of nodes, suggesting that the branching rule increased the complexity of the LP phase.
more than any other rule. This seems to suggest that any possible theoretical benefits of the rule were lost because of the extra effort needed to solve the LP relaxation.

An interesting comparison of branching rules is between the node connection cover and the single path branching rule. As will be discussed in Chapter 8, both rules can be applied directly to the pricing problem and remain tractable. Out of the 34 instances, the node connection cover rule outperformed the single path rule in 15 instances and did worse in 8. This suggests that the node connection cover rule would likely perform better but this is in no way certain.

7.4. Effective Branching Rules

The congested arc branching rule was the only one of our methods that consistently raised its lower bound and avoided the symmetry. This method outperformed all of our other branching rules in virtually every instance. One exception was the metro instance; the rule did not converge, while many of our other branching rules did. This was the only instance on which the congested arc rule performed worse. Curiously, when this instance was run on the 32 bit version of CPLEX the branching rule converged very quickly.

The congested arc rule converged to the same upper bound solution as the default optimizer in every instance and performed as well in all but five instances (the metro instance included). In the bhv4 instance, the rule converged significantly faster than the default optimizer.

7.5. Branching Trees

For each of our branching rules, we selected two instances, toronto and bhv5, to display the branch and bound tree after five minutes of CPU time. We present the graphs in two settings: one with the default parameters (the type used for our results)
and one with node dives and node pre-solve turned off. The idea is that node dives and node pre-solve may skew the shape of the branch and bound tree.

For each branch and bound tree, different scalings were used to try and provide the clearest image of the tree. Different scalings that could be modified for the branch and bound tree include the level, sibling, and subtree separation, as well as the radius of nodes.

During the branch and bound process, CPLEX selects the subproblem with smallest lower bound. For an unbalanced branching rule, one side of the branching tree would likely have a smaller lower bound than the other. As a result, we would expect CPLEX to continuously select that side of the branch and bound tree, as depicted in Figure 7.5.1a. For a balanced branching rule, we would expect a more balanced node selection, resulting in a more compact branch and bound tree. This is depicted in Figure 7.5.1b.

![Figure 7.5.1. Unbalanced and balanced node selection](image-url)
Figure 7.5.2. Toronto: Single Variable

Figure 7.5.3. Toronto: Node Connection Cover
Figure 7.5.4. Toronto: Balanced $st$-cut

Figure 7.5.5. Toronto: Shortest Path $st$-cut
Figure 7.5.6. Toronto: Single Path

Figure 7.5.7. Toronto: Congested Arc Rule
Figure 7.5.8. Toronto: CPLEX Default MIP Solver

Figure 7.5.9. bhv5: Single Variable
Figure 7.5.10. bhv5: Node Connection Cover

Figure 7.5.11. bhv5: Balanced $st$-cut
Figure 7.5.12. bhv5: Shortest Path $st$-cut

Figure 7.5.13. bhv5: Single Path
Figure 7.5.14. bhv5: Congested Arc Rule

Figure 7.5.15. bhv5: CPLEX Default MIP Solver
Further Application to Branch and Price

8.1. Introduction to Branch and Price

Our future research is to implement these branching rules using branch and price. In its simplest form, branch and price is a method that combines two well known procedures in optimization: column generation and branch and bound. However, for branch and price to be implemented properly, certain conditions of the branching process have to be met in order to maintain the structure of the pricing problem.

Branch and price is a process that uses column generation to solve the linear program at every node of the branch and bound tree. In our case, each subproblem is modeled using path variables instead of flow variables. The method involves considering every possible path that could satisfy the origin to destination demand of a commodity as a path variable. Since there could be an exponentially large number of path variables in the model, only a small set of columns are considered where each column corresponds to a path variable. This is called the restricted master problem (RMP). A subproblem for each commodity, called the pricing problem, which can be thought of as a separation problem of the dual LP, must be solved to identify the most attractive path variables (of that commodity). The most attractive path variable is then selected to enter the basis of the RMP. The pricing problem of a commodity is formulated using flow variables, and is subject to the flow conservation constraints corresponding to that commodity. Any feasible solution of the pricing problem, i.e., a set of flow variable satisfying flow conservation, represents a path variable. The most attractive path variable is the one which minimizes the reduced cost. This can
be found with a shortest path algorithm. If the most attractive path of the pricing problem has a negative reduced cost, its corresponding column is added to the RMP and the LP is re-optimized. This process is repeated until no attractive path variables can be found to enter the basis. At this point, the solution is either integer feasible or a branching rule is implemented.

There has been extensive research on developing column generation models for multi-commodity flow problems, see Barnhart et al. [2, 3] and Parker et al. [8]. When solving the LP problem in standard node-arc formulation, a large proportion of the computation effort is used in storing and updating the inverse of the basis matrix. Column generation exploits the structure of the model so that this effort can be greatly reduced.

Successful implementation of branch and price can strengthen the LP relaxation, decrease problem symmetry, and improve the computational processing of the LP solution [2]. Additionally, for the minimum congestion ODIMCFP, a branch and price implementation would eliminate cyclic flow, which should further decrease redundant branches.

The drawback of column generation is the difficulty in branching. Traditional branching methods would involve finding a path variable with fractional flow and setting that variable to zero or one. Let $x^k_p$ denote a path variable with fractional flow from a commodity $k$. One could easily enforce $x^k_p = 1$ in the RMP by not pricing in any new path variables for commodity $k$. However, enforcing $x^k_p = 0$ is very difficult. This constraint would destroy the structure of the pricing problem. This is because there is no guarantee that $p$ will not be the shortest path in the pricing problem. In fact, based on the LP solution, it is very likely that $p$ would be the shortest path of the pricing problem. As a result the pricing problem becomes a next shortest path procedure. As the number of branches increase, the number of next shortest paths needed to calculate the pricing problem would increase.
The branch and price method presented by Barnhart et al. [3] creates a branching rule that is based on the flow variables in the compact formulation of the model. The branching constraints of the rule are directly added to the pricing problem. Branch and price can be implemented in this way if each branching constraint contains flow variables of a single commodity. Otherwise, the structure of the pricing problem would be destroyed. In addition, certain conditions of the branching constraint must be met to maintain tractability in the pricing problem. The pricing problem could then be solved using Dijkstra’s shortest path algorithm. Their branching rule, which our node connection cover rule is based on, demonstrates how this can be accomplished. We will show that our single path branching rule fits these criteria as well.

For our other branching algorithms, we plan on imposing the branching rule by adding constraints to the RMP. The plan is to reformat the branching rule in terms of path variables instead of flow variables. This method lifts the condition that the branching constraint can only consist of one type of commodity. A downside of this method is that it adds negative costs to the flow variables during the pricing phase. This could potentially create negative cycles. There are numerous polynomial algorithms that can find shortest paths when they exist or detect a negative cycle when present [1]. However, if a negative cycle is found the pricing problem would be NP-hard. If this happens, we would have to solve the pricing problem with a less efficient method that would not guarantee tractability. We have yet to verify whether or not a negative cycle will form during the branching process.

8.2. Path Model

To contrast the differences in size of the compact model and the path model, consider a graph $G = (N, A)$ with set of commodities $K$ for any multi-commodity flow problem. The compact formulation would contain a large number of constraints $(|A| + |N| \cdot |K|)$ and a relatively large number of variables $(|A| \cdot |K| + 1)$. On the other
hand, the path formulation would contain a relatively small number of constraints
(|\(K| + |A|\)) and a potentially huge number of path variables. For example, a relatively
dense network of 20 nodes and two commodities could contain well over a million path
variables.

Again consider the graph \(G = (N, A)\) with capacities \(u_{ij}\) for all \((i, j) \in A\), a set of
commodities \(K\) with pairs \((s_k, t_k)\) and quantity \(v_k\) for each \(k \in K\). Let \(P_k\) represent
the set of all origin-destination paths in \(G\) for commodity \(k \in K\). For each path
\(p \in P_k\) there is a path variable \(x^k_p\) that denotes the total percentage of flow from
commodity \(k\) that travels through the entire path \(p\). For an integer program each
commodity travels along a single acyclic path. Thus any feasible solution has one
path variable \(p^* \in P_k\) for each \(k \in K\) such that \(x^k_{p^*} = 1\) and for all other paths,
\(p \in P_k \setminus p^*\), would have \(x^k_p = 0\) for all \(k \in K\). The relaxation of the integer program
allows each path variable to take on all positive real numbers, i.e., \(x^k_p \geq 0\). We
do not restrict \(x^k_p \leq 1\) because the restriction is already enforced by the constraint
\(\sum_{p \in P_k} x^k_p = 1\) for all \(k \in K\). By not adding these constraints, we eliminate a large
number of constraints from the path model. Having as few constraints in the path
model as possible is important because it limits the number of dual variables in the
model, making it easier to calculate the reduced costs of a solution. The column
generation model of the LP can then be formed as:

\[
\min \quad \alpha \\
\sum_{p \in P_k} x^k_p = 1 \quad \forall k \in K \\
u_{ij} \alpha - \sum_{k \in K} v_k \sum_{p \in P_k; (i,j) \in p} x^k_p \geq 0 \quad \forall (i,j) \in A \\
x^k_p \geq 0 \quad \forall p \in P_k, \quad \forall k \in K.
\]

To solve the LP we use a RMP, which leaves out most of the columns from the
model. In fact, the first formulation often starts with only one feasible path variable
for each commodity \( k \in K \). These paths could be found by independently finding a maximum capacity path for each commodity. We then add path variables for each commodity \( k \in K \) if there is a path \( p \in P_k \) with a negative reduced cost. For each commodity \( k \in K \) we find a path \( p \in P_k \) with the minimum reduced cost by solving the \textit{pricing problem} for commodity \( k \). Let \( \pi_k \) denote the dual variable of commodity \( k \) of the flow conservation constraint, i.e., the constraint \( \sum_{p \in P_k} x^k_p = 1 \) and let \( \theta_{ij} \) be the dual variable corresponding to the capacity constraint for arc \((i, j) \in A\). Note that \( \theta_{ij} \geq 0 \) and \( \pi_k \) is unrestricted. The reduced costs for path variable \( x^k_p \) could be calculated:

\[
\text{reduced cost of path variable } x^k_p = \pi_k + \sum_{(i, j) \in A: (i, j) \in p} v_k \theta_{ij}.
\]

The pricing problem is subject to the flow conservation constraints (of the compact model) for a commodity \( k \in K \), thus any feasible solution to the pricing problem is a feasible path of commodity \( k \). A solution of the pricing problem gives the path \( p \in P_k \) with the minimum reduced cost for commodity \( k \). The pricing problem is in terms of flow variables \( f^k_{ij} : (i, j) \in A \) and not in terms of path variables \( x^k_p : p \in P_k \). The reduced cost for path variable \( x^k_p \) can be rewritten in terms of flow variables by having \( f^k_{ij} = 1 \) if \((i, j) \in p\) and \( f^k_{ij} = 0 \) if \((i, j) \not\in p\). The pricing problem for a commodity \( k \in K \) can be formulated in terms of flow variables as:

\[
\min \quad \pi_k + \sum_{(i, j) \in A} v_k \theta_{ij} f^k_{ij}
\]

\[
\sum_{j : (i, j) \in A} f^k_{ij} - \sum_{j : (i, j) \in A} f^k_{ji} = \begin{cases} 
1 & \text{if } i = s_k \\
-1 & \text{if } i = t_k \\
0 & \text{otherwise} 
\end{cases} \quad \forall i \in N
\]

\[
f^k_{ij} \geq 0 \quad \forall (i, j) \in A.
\]
Note that there are no capacity constraints in the model, thus if there is a path from \( s_k \) to \( t_k \) that minimizes cost, then the entire supply from \( s_k \) can travel along that path to \( t_k \). For this reason, it is not necessary to require \( f^k_{ij} \in \{0, 1\} \) and simply having \( f^k_{ij} \geq 0 \) is sufficient. Also, the cost corresponding to the flow through an arc is non negative, i.e., \( v_k \theta_{ij} \geq 0 \) for all \( (i, j) \in A \). Since there are no negative arc costs in the model, the pricing problem can be solved easily with an efficient shortest path algorithm such as Dijkstra’s algorithm.

8.3. Implementing Branch and Price

For our future research, we will implement a branch and price technique in one of two ways depending on the branching rule. We hope to implement these methods in a way that will keep the pricing problem tractable and provide a better partition of the feasible set. It is important to note that implementing these methods using CPLEX or some other optimization software is not a trivial task.

Our first method for branch and price will add the constraints of the branching rule directly to constraints of the pricing problem followed by enumerating a polynomial number of subproblems for the pricing problem. This method is a variation between the methods used by Parker and Ryan [8] and Barnhart et al. [3]. Various conditions need to be met in order to maintain the structure of the pricing problem and to keep the processing time of the pricing problem tractable.

Our second method for branch and price will add the branching constraints to the RMP. This method will involve transforming the branching rule from the compact formulation into the path formulation and then calculating the dual variables of the branching constraints. This method will eliminate some of the conditions needed in the first method but may increase the difficulty of the pricing problem.
8.3.1. Adding Constraints to the Pricing Problem. Our first branch and price method would apply our branching rule directly to the constraints of a pricing problem. In order to implement this method, the set of flow variables in each branching constraint can only belong to one commodity. Let

$$\delta = \{(i_1, j_1, k_1), (i_2, j_2, k_2), \ldots, (i_\mu, j_\mu, k_\mu) : (i_h, j_h) \in A, k_h \in K, h = 1, 2, \ldots, \mu\}$$

be a set of flow variables corresponding to a branching constraint. For the structure of the pricing problem to be maintained, every flow variable in $\delta$ must be of the same commodity, i.e., $k_1 = k_2 = \cdots = k_\mu$. If we tried to add the constraint to the pricing problem and $\delta$ contained flow variables from multiple commodities, the flow variables of the pricing problem would no longer be in terms of a single commodity, hence the structure of the pricing problem would be destroyed. For this reason, the congested arc rule cannot be implemented into the constraints of the pricing problem but our other branching rules could. However, only the node connection cover and the single path branching rules could be implemented in this way and keep the pricing problem tractable.

Consider the single variable branching rule, that creates the branches $f^k_{ij} = 0$ and $f^k_{ij} = 1$, $(i, j) \in A$, over the graph $G = (N, A)$. The first branch can still be solved as a shortest path problem. However, the second branch $f^k_{ij} = 1$ would have to be solved by two shortest path problems, one for the node pair $s_k \rightarrow i$ and another one for $j \rightarrow t_k$. Suppose now that there were multiple branches implemented and the pricing problem had the branching constraints $f^k_{ij} = 1$ and $f^k_{hw} = 1$ and that $i \neq j \neq h \neq w \neq s \neq t_k$. The pricing problem would require six shortest path problems to find an optimal solution. These shortest paths would be $s_k \rightarrow i, j \rightarrow h, w \rightarrow t_k$ and $s_k \rightarrow h, w \rightarrow i, j \rightarrow t_k$. Very quickly we see that there could be an exponentially large number of shortest path problems needed to solve the pricing problem to optimality, hence implementing this branching procedure would render the pricing phase \textit{intractable}. 

68
As discussed earlier the node connection cover rule partitions a set of arcs leaving a node into two sets for some commodity \( k \in K \) with a bifurcation in flow through that node. The two branches then restrict the flow of commodity \( k \) from passing through one of the two partitions. This can easily be enforced in the pricing phase by setting the partition of flow variables to zero and then solving the shortest path problem. This is good because it keeps the pricing problem tractable but has the drawback that it does not provide a true partition of the feasible set.

Our single path rule provides a true partition of the feasible set and would keep the pricing problem tractable. The single path rule has the set of flow variables \( \delta \) form a path \( p \) from \( s_k \) to some specified node \( h \in N \). One of the branches would be \( \sum_{(i,j) \in p} f^k_{ij} \geq |\delta| \). This can easily be enforced in the pricing problem by setting \( f^k_{ij} = 1 \), for all \( f^k_{ij} \in \delta \). The pricing problem for this branch could then be solved as a shortest path problem from \( h \rightarrow t_k \). Any additional branches of this type, on commodity \( k \), would simply be an extension of the path \( p \).

The other branch \( \sum_{(i,j) \in p} f^k_{ij} \leq |\delta| - 1 \) creates a constrained shortest path problem which can be much more difficult to solve. However, because our branching rule is composed of flow variables that form a path originating from the source node, the pricing problem could be solved by creating a polynomial number of subproblems (of the pricing problem) and solving each subproblem as a shortest path problem. Consider a single branch of this type forming path \( p \). Each subproblem would enforce that a single flow variable along the path \( p \) would equal zero, and all flow variables on the arcs preceding that flow variable (along the path \( p \)) would equal one. The result would be \( |\delta| \) subproblems where each subproblem would force flow down the path \( p \) for a distance of 0, 1, 2, ..., or \( |\delta| - 1 \) and not allow flow through the next flow variable along the path. Each subproblem could then be solved as a shortest path problem. These \( |\delta| \) subproblems would contain all possible solutions of \( \sum_{(i,j) \in p} f^k_{ij} \leq |\delta| - 1 \).
Figure 8.3.1 gives a small example of the various subproblems needed in the pricing problem for a single branching constraint of the single path rule. In this example, we have some commodity with origin and destination nodes \((s, t)\). The set \(\delta\) consists of flow variables that form the path \(p: (s, 1) - (1, 2) - (2, 3) - (3, 4)\). In the figure, branch one depicts the pricing problem for the branching constraint \(\sum_{(i,j) \in p} f_{ij}^k \geq 4\), with a double circle denoting the shortest path from \(4 \rightarrow t\). Branch two denotes the four subproblems of the pricing problem that would be needed to enforce \(\sum_{(i,j) \in p} f_{ij}^k \leq 3\). In the diagram, the arrows without an \(\times\) indicate that the flow through the arc is set to one, and an \(\times\) indicates that no flow is allowed through that arc.

For multiple branches on a single commodity, the process of making subproblems would be entirely similar. Consider three branch constraints on a commodity forming paths \(p_1: (s, 1) - (1, 2) - (2, 3) - (3, 4)\), \(p_2: (s, 1) - (1, 2) - (2, 5)\), and \(p_3: (s, 6) - (6, 7) - (7, 8) - (8, 9)\), all with the \(\leq\) constraints. The pricing problem could be found by solving 6 shortest path problems as depicted in Figure 8.3.2.
We strongly believe that if there is a set of $B$ branches corresponding to a commodity $k$ and for each branch $b \in B$ there is a set of flow variables $\delta_b$, then the maximum number of subproblems needed to solve the pricing problem of commodity $k$ would be $|\bigcup_{b \in B} \delta_b|$. Note that $\delta_b$ would never have a flow variable that flows into the source node, thus the union of all branch variables corresponding to a commodity would be strictly less than the total number of arcs. Although we have yet to put together a formal proof, there are strong indications that this is true based on our examples. This means that the number of subproblems needed to solve the pricing problem would always be less than the number of arcs in the graph. Thus this method would have the advantage of remaining tractable while providing a partition of the solution set. From our results in compact formulation, the node connection cover rule performed better, on average, than the single path rule. However, this may not be the case in branch and price.

8.3.2. Adding Constraints to the RMP. We hope to implement our other branching rules by adding the branching constraints of the RMP. This method would provide a more universal method of imposing branching rules based on flow variables. By adding the constraint to the RMP and not the pricing problem, the structure
of the pricing problem would not be changed. Thus any branching rule based on
Theorem 2 could be implemented in branch and price and the pricing phase would
always remain tractable. This means we could implement the congested arc rule, our
strongest method, using this procedure.

Consider any non-integer LP solution with solution \( \bar{f} \) and two sets of flow vari-
ables:

\[
\begin{align*}
\delta_1 &= \{(i_1, j_1, k_1), (i_2, j_2, k_2), \ldots, (i_y, j_y, k_y) : (i_h, j_h) \in A, k_h \in K, h = 1, 2, \ldots, y\} \\
\delta_2 &= \{(g_1, e_1, q_1), (g_2, e_2, q_2), \ldots, (g_w, e_w, q_w) : (g_h, e_h) \in A, q_h \in K, h = 1, 2, \ldots, w\}
\end{align*}
\]
such that the difference in flow of the two sets is non-integer, i.e., \( \sum_{(i,j,k) \in \delta_1} \bar{f}_{ij}^k - \sum_{(i,j,k) \in \delta_2} \bar{f}_{ij}^k = r \notin \mathbb{Z} \). By Theorem 2 we can implement the branching rule in
compact form as:

\[
\begin{align*}
\sum_{(i,j,k) \in \delta_1} f_{ij}^k - \sum_{(i,j,k) \in \delta_2} f_{ij}^k &\geq \lceil r \rceil \\
\sum_{(i,j,k) \in \delta_1} f_{ij}^k - \sum_{(i,j,k) \in \delta_2} f_{ij}^k &\leq \lfloor r \rfloor.
\end{align*}
\]

Our next step is to format the rule in terms of path variables. Consider the branch
\( \sum_{f \in \delta_1} f - \sum_{f \in \delta_2} f \geq \lceil r \rceil \) added as a constraint to the RMP. The coefficient for each
path variable would be the difference in the number of flow variables that the path
shares with the sets \( \delta_1 \) and \( \delta_2 \). Thus, if a path \( p \in P_k \) had three flow variables in \( \delta_1 \)
and one flow variable in \( \delta_2 \) then the path variable \( x_{p}^k \) would have a coefficient of 2 for
the constraint. The branching rule, in the path formulation, would be:

\[
\begin{align*}
\sum_{k \in K} \sum_{p : p \in P_k} (|p \cap \delta_1| - |p \cap \delta_2|) x_{p}^k &\geq \lceil r \rceil \\
\sum_{k \in K} \sum_{p : p \in P_k} (|p \cap \delta_1| - |p \cap \delta_2|) x_{p}^k &\leq \lfloor r \rfloor.
\end{align*}
\]
The reduced costs of a path variable $x^k_p$, $p \in P_k$, could change if $p \in \delta_1 \cup \delta_2$, thus a new reduced cost would have to be calculated for the path variables. Let $B$ denote the set of all branch constraints added to the RMP, and $\gamma_b$: $b \in B$ denote the dual variable of the branching constraint $b$. Let $\delta^b_1$, $\delta^b_2$ denote the two sets of flow variables corresponding to branch $b \in B$ and let $r_b$ be either the rounded up or rounded down difference of $\delta^b_1$ and $\delta^b_2$ (it will be one of the two depending on the branch). Note that by properties of duality $\gamma \geq 0$ for the branch $\sum_{k \in K} \sum_{p \in P_k} (|p \cap \delta_1| - |p \cap \delta_2|) x^k_p \geq \lceil r \rceil$ and that $\gamma \leq 0$ for the branch $\sum_{k \in K} \sum_{p \in P_k} (|p \cap \delta_1| - |p \cap \delta_2|) x^k_p \leq \lfloor r \rfloor$. The reduced cost of path variable $x^k_p$ could then be calculated as:

$$\text{reduced cost of } x^k_p = \pi_k + \sum_{(i,j) \in A: (i,j) \in p} v_k \theta_{ij} + \sum_{b \in B} (|p \cap \delta^b_1| - |p \cap \delta^b_2|) \gamma_b$$

As before, we have to translate the reduced cost in terms of flow variables, so that we can solve the pricing problem in terms of flow variables. Let $I_{f^k_{ij} \in \delta^b_1}$ be an indicator function with a value one if $f^k_{ij} \in \delta^b_1$ and zero otherwise, and analogously for $I_{f^k_{ij} \in \delta^b_2}$. Then the coefficient corresponding to the cost of flow variable $f^k_{ij}$ in the pricing problem of commodity $k$ would be $v_k \theta_{ij} + \sum_{b \in B} \gamma_b (I_{f^k_{ij} \in \delta^b_1} - I_{f^k_{ij} \in \delta^b_2})$. The pricing problem for commodity $k$ could be modeled as:

$$\min \pi_k + \sum_{(i,j) \in A} \left( v_k \theta_{ij} + \sum_{b \in B} \gamma_b (I_{f^k_{ij} \in \delta^b_1} - I_{f^k_{ij} \in \delta^b_2}) \right) f^k_{ij}$$

$$\sum_{j \in A: (i,j) \in A} f^k_{ij} - \sum_{j \in A: (j,i) \in A} f^k_{ji} = \begin{cases} 1 & \text{if } i = s_k \\ -1 & \text{if } i = t_k \forall i \in N \\ 0 & \text{otherwise} \end{cases}$$

$$f^k_{ij} \geq 0 \forall (i,j) \in A.$$
Note that the structure of the pricing problem is preserved and the cost of each flow variable could take on either positive or negative values depending on the branch dual variables. The negative costs may result in negative cycles. For this reason, we use a shortest path problem with cycle detection. If there are no negative cycles, paths could be found to enter the basis of the RMP using this algorithm and the pricing phase would remain tractable. However if there was a negative cycle, the pricing problem would be $NP$-hard and we would have to find a different method for solving the pricing problem.

8.4. Final Remarks for Branch and Price

The future applications of these branching strategies are promising. We have devised two methods in which we could implement our branching rules. The first method adds constraints directly to the pricing problem. We have devised a new branching rule, the single path rule, which can be applied in this way and keep the pricing problem tractable. Our second method of branch and price gives a more general approach by adding constraints directly to the RMP. This second approach has the advantage of allowing the branching constraints to be composed of any two sets of flow variables, thus allowing use to use branching rules that were much more effective in the compact formulation. The disadvantage of this method is that the pricing problem may become intractable if a negative cycle forms.

In general, by structuring the model with a huge number of path variables and applying a branch and price method we hope to improve numerous difficulties that were encountered in the compact model. Some of these improvements include:

- reducing the computation effort for solving the LP by decreasing the number of constraints of the model;
- eliminating cyclic flow by using only path variables, thereby reducing redundant branches;
• reducing the amount of symmetry by representing the model in terms of a large number of variables.
CHAPTER 9

Conclusions

In this paper we have presented branching rules for the minimum congestion origin-destination integer multi-commodity flow problem. We have provided six branch and bound rules for this problem and provided computational and graphical results of the rules. Additionally, we have provided a foundation for two ways of implementing our rules with branch and price using a path generation model.

We found that our branching rules that were based on flow variables of a single commodity resulted in numerous redundant branches. There was often little change in the objective function throughout the branching process. However, adding cuts after the branching phase may help reduce these redundancies as demonstrated in Barnhart et al. [3].

Our congested arc branching rule found the most congested arc with infeasible flow and forced either more or less flow (of any type) through that arc. This created a much more efficient branching rule than any of our single commodity rules. With a few exceptions, this method performed as well as the CPLEX default MIP solver and even outperformed the default solver in one instance.

The first method that we presented for branch and price was to add the branching constraint directly to the pricing problem. For this method, the branching rule must be based on a single commodity (to retain the structure of the pricing problem) and certain conditions have to be met to remain tractable. We presented two methods that meet these criterion: the node connection cover rule, from Barnhart et al. [3], and our own method, the single path path rule.
The second method that we presented for implementing branch and price was by adding constraints to the restricted master problem (RMP). By adding constraints to the RMP, we would not need the single commodity restrictions that were required when adding the constraints to the pricing problem. For this method, we demonstrated how our branching constraints, which are formulated from the flow variables in compact formulation, can be transformed into path variable constraints. The downside of this method is that it would add negative costs in the pricing problem. This could potentially render the pricing problem intractable if a negative cycle forms.
Bibliography


