University of Nebraska at Omaha
DigitalCommons@UNO

Mathematics Theses, Dissertations, Research and Student Creative Activity

Department of Mathematics

5-2024

The Double-Pivot Network Simplex Method

Jordan Mae Sahs
University of Nebraska at Omaha, jsahs@unomaha.edu

Follow this and additional works at: https://digitalcommons.unomaha.edu/mathstudent

Please take our feedback survey at: https://unomaha.az1.qualtrics.com/jfe/form/SV_8cchtFmpDyGfBLE

Recommended Citation

Sahs, Jordan Mae, "The Double-Pivot Network Simplex Method" (2024). *Mathematics Theses, Dissertations, Research and Student Creative Activity*. 3.
https://digitalcommons.unomaha.edu/mathstudent/3

This Thesis is brought to you for free and open access by the Department of Mathematics at DigitalCommons@UNO. It has been accepted for inclusion in Mathematics Theses, Dissertations, Research and Student Creative Activity by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.

The Double-Pivot Network Simplex Method

A Thesis presented to the

Department of Mathematical and Statistical Sciences

and the

Faculty of the Graduate College
University of Nebraska

in partial fulfillment
of the requirements of the degree

Master of Arts

University of Nebraska at Omaha

by

Jordan M. Sahs

May 2024

Supervisory Committee

Dr. Fabio Torres Vitor

Dr. Betty Love

Dr. Hesham Ali

The Double-Pivot Network Simplex Method

Jordan M. Sahs, MA

University of Nebraska, 2024

Advisor: Dr. Fabio Torres Vitor

Abstract

The network simplex method, a minimum-cost network flow algorithm, was first created in 1956 by George Dantzig to solve transportation problems. This thesis improves upon Dantzig's method by pivoting two arcs instead of one at each iteration. The proposed algorithm is called the double-pivot network simplex method. Both leaving arcs are determined by solving a two-variable linear program. Due to the structure of these two-variable problems, this thesis also presents an approach to quickly solve them. The network and double-pivot network simplex methods make use of a modified eXtended Threading Index technique to efficiently create cycles and maintain the spanning tree basis. Computational experiments showed that the double-pivot network simplex method solved minimum-cost network flow problems from the NETGEN benchmark library using approximately 50% fewer total iterations, on average, than the network simplex method. In regards to CPU time, the double-pivoting method outperformed the network simplex algorithm by about 12% in large NETGEN instances. The network simplex method solved smaller NETGEN instances faster than the double-pivot network simplex method by approximately 8%. When averaging all instances, the double-pivoting method proposed in this thesis is over 5% faster than the network simplex method.

©2024 – Jordan M. Sahs
all rights reserved.

Dedicated to my grandma MaryLue McClane and close friends Brad Tuttle & Brad Horner, who are my biggest supporters & role models.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The network simplex method, a minimum-cost network flow algorithm, was first created in 1956 by George Dantzig to solve transportation problems[20]. This algorithm solves minimum-cost network flow problems significantly faster than the simplex method he first developed in 1947 to solve linear programs[9], and was tested to solve transportation problems in 1951[10]. This thesis improves upon Dantzig's network method by pivoting two arcs instead of one at each iteration. This new approach is called the double-pivot network simplex method.

For simplicity, the remainder of this thesis denotes the network simplex method as the *single-pivot network simplex method*. The minimum-cost network flow problem is a mathematical optimization problem that aims to minimize the cost of sending a certain amount of resources (*supply*) to various destinations (*demand*) in a network. The supply and demand locations are represented by nodes, and the pathways that connect them are represented by directed arcs. Additional nodes represent transshipment points at which flow can be aggregated and re-distributed. Arcs may have an upper bound that limits the amount of resources that can traverse them, and there may even exist a lower bound. However, most presentations of the minimum-cost network flow problem do not directly deal with this, as pre-solving with lower bound substitution simplifies the problem.

Various real-world applications of the minimum-cost network flow problem exist. Some of them include solving currency distributions in Kegalle, Sri Lanka[6], patrol coverage for law enforcement to monitor highway locations with frequent vehicular accidents[11], inventory transshipment in airline cargo management[39], emergency fleet management in disastrous situations[7], RNA secondary-structure prediction[15], and optimizing fire emergency routes in coal mines[24]. The demand for fast, reliable network optimization methods to solve continually arising real-world problems remains.

The single-pivot network simplex method is a specialized variant of the simplex method. The simplex method moves from one basic feasible solution to another basic feasible solution at each iteration until an optimal solution is found. While the simplex method can solve minimum-cost network flow problems if presented as a linear program, the single-pivot network simplex method is faster than the simplex method because it preserves the graphical aspects of the original problem by maintaining the basis as a spanning tree. Overall, the single-pivot network simplex method starts with a spanning tree as a solution. The algorithm selects one entering arc and adds to the spanning tree, thus creating a cycle. A leaving arc is then determined and removed from the solution to maintain the spanning tree structure, and the process is repeated until an optimal solution is found.

One of the most time-consuming steps of the single-pivot network simplex method is how the spanning tree basis is stored, maintained, and updated. Glover et al. in

1973 developed the *Augmented Threaded Index* method, which is one of the most efficient data labeling procedures for the single-pivot network simplex method[14]. In the late 1990s, Orlin[26] and Tarjan[30] also deployed various improving methods to the single-pivot network simplex method, including how the data structure within the spanning tree basis is maintained. Other research on the efficiency of the single-pivot network simplex method also exists[38 26 27 1].

One should notice that most optimization methods are considered one-dimensional search algorithms. That is, at each iteration an improved solution is obtained by moving along a single direction[32]. However, multidimensional search methods have also been explored, and aim to solve optimization problems faster than the traditional one-dimensional search methods via searching over multiple directions at each iteration instead of one.

Karmarkar and Ramakrishnan discussed the first presentation of a multidimensional search algorithm in 1985[19]. Since then, various multidimensional methods have been developed to optimize a wide range of problems. This includes the double-pivot simplex method from Vitor and Easton[33], their two-dimensional search interior point algorithms[34 36 35], Vitor's ratio algorithm for two-constraint linear programs[31], the two and three-dimensional search interior point methods of Boggs et al.[5] and Domich et al.[12], Santos et al.'s optimization methods for interior point algorithms[28], and Yang and Vitor's double-pivot degenerate-robust simplex algorithm[37].

Given the potential of multidimensional search methods to solve optimization problems faster than their corresponding one-dimensional search techniques, this thesis creates the first multidimensional search algorithm to solve minimum-cost network flow problems if presented as a network. Instead of exchanging one arc in the spanning tree basis at each iteration, the double-pivot network simplex method can exchange up to two arcs at a time. Both leaving arcs are determined by solving a two-variable linear program, and this thesis also presents a quick technique to solve these subproblems. This thesis also compares the computational efficiency of the double-pivot network simplex method versus the single-pivot network simplex method. Both algorithms were implemented using Barr et al.'s data labeling procedure[2].

Comparisons of the single and double-pivoting methods are measured by average CPU runtime and the number of pivots performed on a set of benchmark problems from the NETGEN test suite stored by Kovács[23]. Computational experiments showed that the double-pivot network simplex method solved some benchmark minimum-cost network flow problems using approximately 50% fewer iterations, on average, than the single-pivot network simplex method. Regarding CPU time, the double-pivot network simplex method solved larger benchmark instances by more than 12% faster than the single-pivot network simplex method. Conversely, the single-pivoting method solved smaller benchmark problems around 8% faster than the double-pivoting technique. When averaged, the double-pivot network simplex method is more than 5% faster than the single-pivoting method.

The remainder of this thesis is organized as follows. Chapter 2 presents the necessary background information for the reader to understand the single-pivot network simplex method. This includes the core definitions and formulations of the minimum-cost network flow problem, various pivoting rules that reduce the per-iteration computation times for entering arcs, the aforementioned enhanced labeling procedures by Barr et al. to reduce the number of calculations required to maintain the spanning tree basis, a summary on how to implement the single-pivoting method, and an example.

Chapter 3 describes the theoretical and algorithmic results of the double-pivoting method, and presents an approach to determine the change of flow when two arcs are exchanged in the spanning tree basis at a time. The section introduces the core definitions and casework needed to understand the complications that arise when introducing two cycles into the spanning tree basis, a two-variable problem to account for the change of flow of overlapping cycles, and enhanced labeling procedures to reduce the overall runtime needed to calculate overlapping cycles and flow. It concludes with an implementation summary of the double-pivoting method, and an example.

Chapter 4 presents the results of computational experiments performed to test both pivoting methods on the NETGEN benchmark instances. The computational study also compares against open-source and commercial mathematical programming solvers. Implementation details are also discussed. Lastly, Chapter 5 concludes the thesis and presents future research topics that can further advance the double-pivot network simplex method.

2 Background Information

This chapter presents the necessary background information needed to understand this thesis. The chapter discusses the formulation of the minimum-cost network flow problem as well as the single-pivot network simplex method, the block pivoting rule[21], Barr et al.'s enhanced spanning tree labeling procedure, the complete steps of the single-pivot network simplex method, and a numerical example.

2.1 The Minimum-Cost Network Flow Problem

Formally, let $G = (N, A)$ be a graph where N denotes the set of nodes and A represents the set of arcs. The minimum-cost network flow problem can be formulated as the following linear program:

$$\text{Minimize: } z = \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{Subject to: } \sum_{(i,n)\in A} x_{in} - \sum_{(n,j)\in A} x_{nj} = b_n \quad \forall n \in N$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A$$

where u_{ij} denotes the upper-bound of each arc $(i, j) \in A$, c_{ij} denotes the cost per unit of flow sent on every arc $(i, j) \in A$, and b_n is the demand ($b_n \leq 0$) or supply ($b_n > 0$)

of each node $n \in N$. The sole constraint represents the notion that flow must be conserved. That is, the flow that goes into every node must go out. To demonstrate the minimum-cost network flow problem, consider **Example** 1 where b_n denotes the supply/demands of each node $n \in N$, and the costs and upper bounds appear in the form (c, u) above the arcs.

Example 1 *Suppose there exist six warehouses with several delivery trucks. The problem seeks to transport goods between these warehouses minimizing the total cost. The supply/demand for each warehouse, the transportation cost, and the number of goods that can be transported between roads due to the truck's carrying capacity are shown in* ***Figure*** *2.1.*

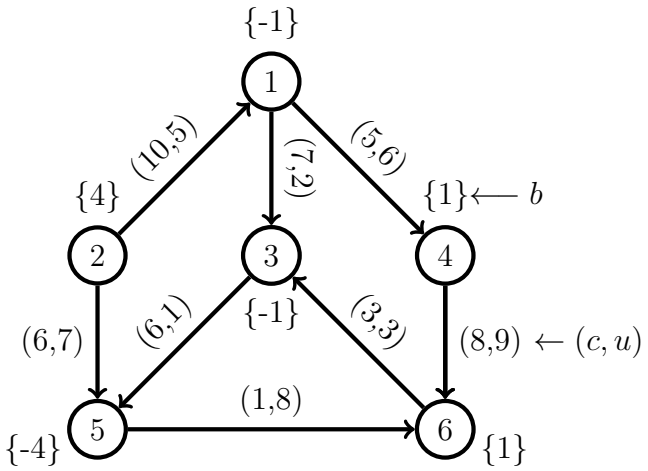


Figure 2.1: Network of **Example** 1.

Notice that **Example** 1 can be modeled as the following linear program and solved using the simplex method[9]:

$$
\begin{aligned}
\text{Minimize:} \quad & z = 7x_{13} + 5x_{14} + 10x_{21} + 6x_{25} + 6x_{35} + 8x_{46} + x_{56} + 3x_{63} \\
\text{Subject to:} \quad & x_{13} + x_{14} - x_{21} = -1 \\
& x_{21} + x_{25} = 4 \\
& -x_{13} + x_{35} - x_{63} = -1 \\
& -x_{14} + x_{46} = 1 \\
& -x_{25} - x_{35} + x_{56} = -4 \\
& -x_{46} - x_{56} + x_{63} = 1 \\
& 0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A.
\end{aligned}
$$

However, the single-pivot network simplex method is a more effective technique since it preserves all the graphical aspects of the problem. The following sections discusses the single-pivoting method in details.

2.2 The Single-Pivot Network Simplex Method

The single-pivot network simplex method uses a spanning tree to represent each basis, also called the *spanning tree basis*. Arcs in this spanning tree are referred to as *basic arcs*. Likewise, the term *non-basic arcs* denotes those arcs that do not appear in the spanning tree. The flow on a non-basic arc is either zero or equal to its upper bound. For convenience, this thesis implements only the Big-M method for generating

an initial basic feasible solution[8 3]. However, other methods exist, such as the Two-Phase method[3].

The initial spanning tree basis is obtained by introducing an artificial node into the network and connecting artificial arcs from all supply and demand nodes to the artificial node. If $b_i > 0$ for a node i, the arc points toward the artificial node from node i. If $b_i \leq 0$, the arc points from the artificial node toward the node i. The spanning tree basis should only contain n arcs where n is the number of nodes in N excluding the artificial node. The artificial node has a demand/supply of zero, and artificial arcs have no upper bound and are not included in the set of arcs A. The per-unit cost of using any of the arcs from the supply nodes is denoted as M where,

$$M = \sum_{(i,j)\in A} |c_{ij}|$$

If $c_{ij} = 0 \ \forall (i,j) \in A$, then M is set to one. This technique prevents skewing the algorithm in the number of iterations it takes to solve if artificial arc costs are too high. Lastly, a basic *root arc* with zero flow is appended to the tree such that the constraint matrix is full rank. In this thesis, the root arc is attached to the *root node* of the spanning tree and is represented as a black square. An initial spanning tree basis to **Example** 1 is presented in **Figure** 2.2 with the tree being rooted on the artificially introduced node seven.

Given an initial spanning tree basis, the single-pivot network simplex method arbitrarily chooses a node and make it the "root" of the spanning tree, and set its dual

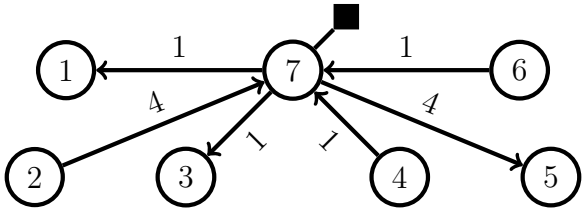


Figure 2.2: Initial spanning tree of **Example** 1.

variable w_i, referred to as *node potential*, equal to zero. The algorithm thus computes the dual variables of all other nodes by $w_i - w_j = c_{ij}$ where i and j are indices denoting nodes that appear in the (i, j) arc structure. The single-pivot network simplex method then computes c^{π}_{ij}, referred to as the *reduced cost*, of all non-basic arcs using the equation $c^{\pi}_{ij} = w_i - w_j - c_{ij}$.

If there exists a $c^{\pi}_{ij} > 0$ for a non-basic arc at its lower bound or a $c^{\pi}_{ij} < 0$ for a non-basic arc at its upper bound, the algorithm introduces this non-basic arc to the spanning tree basis. Consequently, a cycle in the spanning tree is created. Typically, this non-basic arc is referred to as the *entering* arc and also an *improving arc*. If the entering arc is on its lower bound, then the algorithm finds within the cycle the minimum between x_{ij} of all arcs in the opposite direction of the entering arc and $u_{ij} - x_{ij}$ of all arcs in the same direction of the entering arc. If the entering arc is on its upper bound, then the single-pivot network simplex method finds the minimum between x_{ij} of all arcs in the same direction of the entering arc and $u_{ij} - x_{ij}$ of all arcs in the opposite direction of the entering arc.

To demonstrate the change of flow in every one of these cases, consider the cycle presented in the network shown in **Figure** 2.3. For this example, $(7, 11)$ is the arc entering on its lower bound.

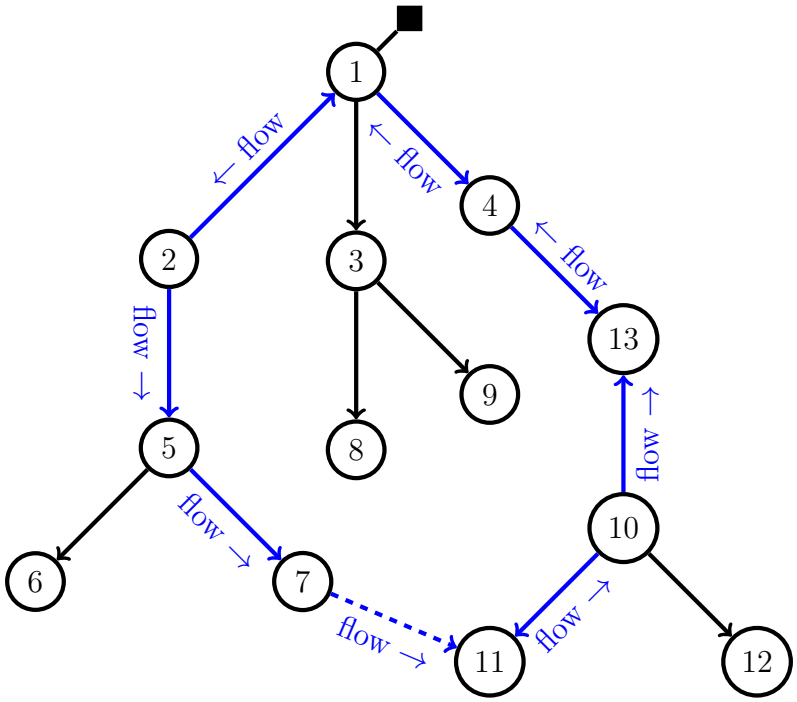


Figure 2.3: Example of a cycle in a spanning tree basis.

Let Δf denote the allowable change of flow for all the arcs within a cycle. The following shows the change of flow for all the arcs within the cycle from **Figure** 2.3:

$$(7, 11) : u_{(7,11)} \quad (10, 11) : x_{(10,11)} \quad (10, 13) : u_{(10,13)} - x_{(10,13)} \quad (4, 13) : x_{(4,13)}$$

$$(1, 4) : x_{(1,4)} \quad (2, 1) : x_{(2,1)} \quad (2, 5) : u_{(2,5)} - x_{(2,5)} \quad (5, 7) : u_{(5,7)} - x_{(5,7)}.$$

Therefore,

$$\Delta f = \min\{u_{(7,11)}, x_{(10,11)}, u_{(10,13)} - x_{(10,13)}, x_{(4,13)}, x_{(1,4)}, x_{(2,1)}, u_{(2,5)} - x_{(2,5)}, u_{(5,7)} - x_{(5,7)}\}.$$

If the entering arc is on its lower bound, $x_{ij} = 0$, the single-pivot network simplex method updates the flow of all the arcs within the cycle in the amount of Δf where arcs in the same direction of the entering arc are decreased and arcs in the opposite direction of the entering arc are increased. If the entering arc is on its upper bound, $x_{ij} = u_{ij}$, then the algorithm updates all the arcs within the cycle in the amount of Δf where arcs in the same direction of the entering arc are decreased and arcs in the opposite direction of the entering arc are increased.

The single-pivot network simplex method then updates all node potentials and all reduced costs for all nodes and arcs, respectively. The process repeats until an arc with an improving c_{ij}^{π} no longer exists. In this case, the algorithm returns the optimal minimum-cost $z^* = \sum_{(i,j)\in A} c_{ij}x_{ij}$ and the optimal spanning tree basis.

One of the bottlenecks of the single-pivot network simplex method is computing the reduced cost for all the arcs to determine the most improving arc[16]. One technique used to ease this computational effort is the block pivoting. The following section discusses this pivoting rule.

2.3 Block Pivoting

Efficient implementations of the single-pivot network simplex method find the most improving arc at each iteration from a selected pool of non-basic arcs instead of computing the reduced cost of all non-basic arcs. Király and Kovács show that block pivoting outperforms most other pivoting rules if the block size is $\sqrt{m}$ where m is the total number of arcs in A excluding the artificial arcs[21]. Király and Kovács also discuss other pivoting techniques, such as candidate list pivoting, alternating candidate pivoting, first eligible arc pivoting, and best eligible arc pivoting. The authors also benchmark each pivoting rule against the NETGEN[23] problem suit and found that block pivoting and alternating candidate pivoting tend to produce the best results.

The *Block Search* method operates by partitioning the non-basic arcs into blocks of a particular size using multiple tracking indices for the arc list and computing the reduced costs of the arcs between those indices. If no improving arcs are found within a block, the algorithm moves on to the next available block and repeats until an improving arc is found, or it iterates through all arcs in every block and terminates.

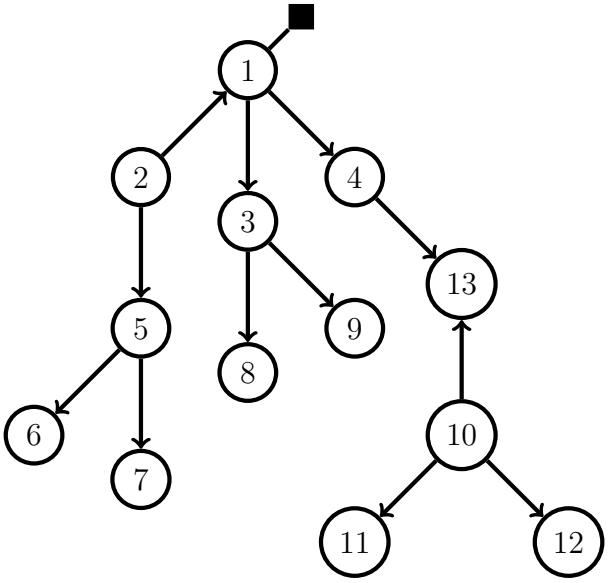
Another important implementation aspect of the single-pivot network simplex method is how the spanning tree basis is maintained and updated throughout the basis exchange process. The following section discusses the procedure used in this thesis.

2.4 The XTI Method

Barr et al. proposed a basis exchange method in 1979 called the XTI (eXtended Threaded Index) method, in which the spanning tree basis can be represented as four lists of length n where n is the total number of nodes in N. The advantage of this approach is that it allows for updating node potentials in a minimal number of steps and quick cycle construction, thereby reducing the total runtime required to update and modify the spanning tree basis. The following notation is used to describe various properties of the spanning tree basis for the XTI method:

$$
\begin{aligned}
p(i) &= \text{predecessor of node } i \text{ where } p(n) = 0 \text{ if } i \text{ is the root node,} \\
s(i) &= \text{number of child nodes of node } i \text{ including itself,} \\
T(q) &= \text{subtree rooted at node } q, \\
T - T(q) &= \text{subtree rooted on the root node, not including the nodes in } T(q), \\
e(i) &= \text{bottom-right, deepest ending node of the subtree rooted at node } i, \\
t(i) &= \text{thread of node } i \text{ (see Section 2.4.3 for more details),} \\
(l, q) &= \text{the leaving arc where } l \text{ is the predecessor of } q, \\
(u, v) &= \text{the entering arc where } v \text{ is the node contained in the predecessor} \\
&\qquad \text{path from } q \text{ to the root node.}
\end{aligned}
$$

Each function p, s, e, and t reference list elements whose indices, i, refer to the node at that index within the list. An example is given in **Figure** 2.4.



n	p	s	e	t	rt	d
1	0	13	12	2	12	0
2	1	4	7	5	7	T
3	1	3	9	8	1	F
4	1	5	12	13	9	F
5	2	3	7	6	2	F
6	5	1	6	7	5	F
7	5	1	7	3	6	F
8	3	1	8	9	3	F
9	3	1	9	4	8	F
10	13	3	12	11	13	T
11	10	1	11	12	10	F
12	10	1	12	1	11	F

Figure 2.4: Example of the XTI method used to document a spanning tree basis.

To facilitate the implementation of the XTI method, the function $d(i)$ was introduced in this thesis where

$$d(i) = \begin{cases} True & \text{if the arc at node } i \text{ points towards the root node} \\ False & \text{if the arc at node } i \text{ points away the root node} \\ 0 & \text{if } i \text{ is the root node.} \end{cases}$$

Using this function, one can quickly build the arcs of any node n using the predecessor path where the arc is $(n, p(n))$ if $d(n)$ is true or $(p(n), n)$ if $d(n)$ is false. In addition,

$$rt(i) = \text{reverse thread of node } i \text{ (see Section 2.4.3 for more details)}$$

is presented as a function in this thesis instead of a procedure as in the original version of the XTI method[2].

2.4.1 Cycle Construction

Using the predecessor and successor functions, one can generate a cycle between nodes u and v (where v is the node contained in the predecessor path from q to the root node of the entering arc), traversing the predecessor paths of both nodes until an intersecting node is found. Once this intersecting node is discovered, the entire cycle has been traversed. The method to find a cycle is shown in **Algorithm** 1. Observe that **Algorithm** 1 assumes arcs are being collected while traversing the paths using the direction function $d(i)$ to generate arcs.

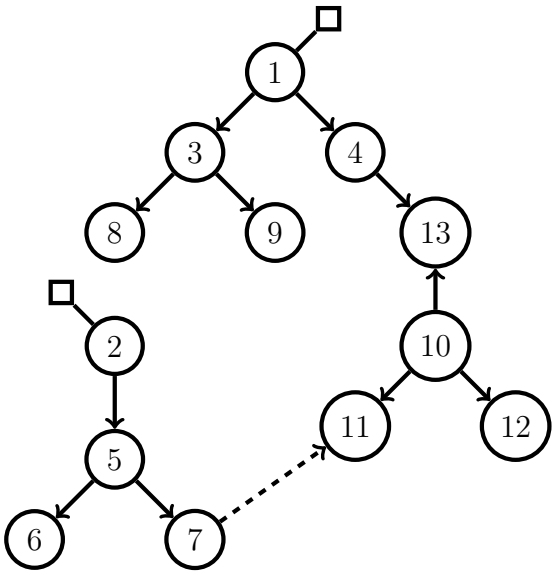
Algorithm 1 XTI method to find a cycle using the predecessor path and successors function.

```
1: procedure FIND CYCLE(u, v, p, s)
2:     Let i ← u
3:     Let j ← v
4:     while i ≠ j do
5:         if s(i) > s(j) then
6:             j ← p(j)
7:         else if s(i) < s(j) then
8:             i ← p(i)
9:         else
10:            j ← p(j)
11:            i ← p(i)
12:        end if
13:    end while
14: end procedure
```

2.4.2 Sub-trees $T(q)$ and $T - T(q)$

To start the basis exchange, the XTI method splits the spanning tree basis into two smaller sub-trees: one rooted on the q node (where q is a part of the leaving arc with l being its predecessor) and the other being all the nodes not in the q subtree. For example, consider the spanning tree basis presented in **Figure** 2.3. The entering arc is $(7, 11)$. Suppose $(2, 1)$ is the leaving arc. Since $q = 2$, the spanning tree basis is split into two sub-trees using the leaving arc as presented in **Figure** 2.5.

Algorithm 2 demonstrates this splitting process. The XTI method finds the smaller subtree to reverse and reattach onto the larger subtree, and it does this by comparing the number of successors in the subtree rooted at q and the subtree $T - T(q)$. This is done so that the fewest nodes are updated. **Algorithm** 3 determines the smaller subtree. Once the smaller subtree is determined, its root is denoted as x_2. The threaded path between x_2 and y_2 must be traversed to update all the node potentials within the subtree.



n	p	s	e	t	rt	d
1	0	9	12	3	12	0
2	0	4	7	5	7	0
3	1	3	9	8	1	F
4	1	5	12	13	9	F
5	2	3	7	6	2	F
6	5	1	6	7	5	F
7	5	1	7	2	6	F
8	3	1	8	9	3	F
9	3	1	9	4	8	F
10	13	3	12	11	13	T
11	10	1	11	12	10	F
12	10	1	12	1	11	F

Figure 2.5: Example of the subtree split operation where $q = 2$.

2.4.3 Threading and Updating Node Potentials

Threading can be described as turning the spanning tree basis into a continuous "beaded necklace". It is the process of traversing left-to-right on this necklace, and reverse threading is the process of traversing right-to-left to temporarily ignore the depth of the tree to visit nodes. A thread is constructed by traversing the tree in a top-down, left-to-right process. For example, the threaded array of **Figure** 2.2 is $t = [2, 3, 4, 5, 6, 7, 1]$ and the reverse thread is $rt = [7, 1, 2, 3, 4, 5, 6]$. This is graphically presented in **Figure** 2.6.

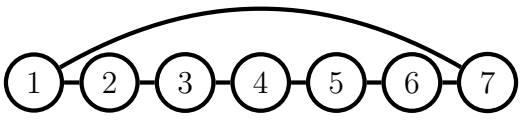


Figure 2.6: Threading diagram of the spanning tree basis depicted in **Figure** 2.2.

Algorithm 2 XTI method to split $T(q)$ and $T - T(q)$ into separate subtrees.

```
1: procedure SPLIT SUBTREES(rt, t, e, s, d, root_node)
2:     Let y ← rt(q)                                   ▷ Updates T − T(q)
3:     t(y) ← t(e(q)).
4:     rt(t(e(q))) ← y.
5:     Let x* ← p(t(e(q))
6:     if x* = 0 then
7:         x* ← root_node
8:     end if
9:     for every node i on the path from p to x* do
10:        e(i) ← y
11:    end for
12:    for every node i on the path from p to root_node do
13:        s(i) ← s(i) − s(q)
14:    end for
15:    p(q) ← 0                                        ▷ Updates T(q)
16:    t(e(q)) ← q
17:    rt(q) ← e(q)
18:    d(q) ← 0
19: end procedure
```

When a sub-tree is rooted at q, this process is "cutting the strings" of the threaded path. Suppose for the spanning tree basis shown in **Figure** 2.2 that the entering arc is $(2, 5)$ and the leaving arc is $(2, 7)$ such that $q = 2$. Therefore, a sub-tree $T(2)$ is created, and the threading diagram in **Figure** 2.7 is formed.

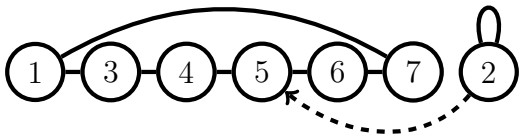


Figure 2.7: Split diagram of the threading diagram depicted in **Figure** 2.6.

Node potentials, w_i, of the smaller subtree on the threaded path from x_2 to y_2 are updated using **Algorithm** 4 for each node i. Once this procedure is finished, the smaller subtree must be reversed and reattached to the larger subtree to finalize the basis exchange.

Algorithm 3 XTI method to determine the smaller subtree between $T(q)$ and $T - T(q)$.

```
1: procedure DETERMINE SMALLER TREE(u,v,q, root_node)
2:     Let x_1 ← q
3:     Let x_2 ← root_node.
4:     Let y_1 ← v
5:     Let y_2 ← u.
6:     if s(x_1) ≤ s(x_2) then
7:         x_1, y_1, x_2, y_2 ← x_2, y_2, x_1, y_1
8:     end if
9: end procedure
```

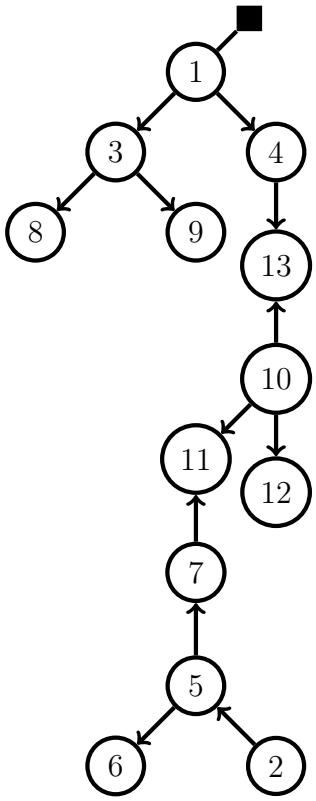
Algorithm 4 Node potential updating procedure for the threaded path.

```
1: procedure UPDATE NODE POTENTIALS(entering arc, x_2, y_2)
2:     Let c_piv ← c^π_ij of the entering arc.
3:     if y_2 equals the "from" node of the entering arc then
4:         c_piv ← −c_piv
5:     end if
6:     w[x_2] ← w[x_2] + c_piv
7:     x ← t[x_2]
8:     while x ≠ x_2 do
9:         w[x] ← w[x] + c_piv
10:        x ← t[x]
11:    end while
12: end procedure
```

2.4.4 Reversing and Reattaching the x_2 Sub-Tree.

The XTI method reverses the smaller subtree to turn y_2 into its new root in preparation for attaching y_2 to y_1. The process of reversing this tree has been greatly improved from the original XTI presentation, inspired by NetworkX's XTI implementation[17], in that instead of doing multiple looping procedures, it can be reduced to one. This is done by treating every level of the subtree as its own smaller subtree and reversing it upwards the predecessor path. **Algorithm** 5 presents the reversing procedure.

Reattaching the subtree, rooted on y_2, back onto the x_1 subtree via y_2 is done in **Algorithm** 6. After the reattaching process, the final spanning tree basis is reattached to the x_1 subtree as shown in **Figure** 2.8 concluding the basis exchange procedure.



n	p	s	e	t	rt	d
1	0	9	12	3	12	0
2	0	4	7	5	7	T
3	1	3	9	8	1	F
4	1	5	12	13	9	F
5	2	3	7	6	2	T
6	5	1	6	7	5	F
7	5	1	7	2	6	T
8	3	1	8	9	3	F
9	3	1	9	4	8	F
10	13	3	12	11	13	T
11	10	1	11	12	10	F
12	10	1	12	1	11	F

Figure 2.8: The new spanning tree basis after the reattachment process is complete.

Algorithm 5 Reformed XTI method for reversing the subtree.

```
1: procedure REVERSE SUBTREE(rt, t, e, p, s, x_2, y_2)
2:     if y_2 ≠ x_2 then
3:         Let P ← a list of all the nodes on the reverse predecessor path y_2 to x_2.
4:         Let Q ← a list of all nodes on the reverse predecessor path p(y_2) to x_2.
5:         for every node i in P and node j in Q do
6:             Let s_i ← s(i)
7:             Let e_i ← e(i)
8:             Let rt_j ← rt(j)
9:             Let e_j ← e(j)
10:            Let t_j ← t(j)
11:
12:            d(i) ← ¬d(j).
13:            p(i) ← j.
14:            p(j) ← 0.
15:            s(i) ← s(i) − s(j).
16:            s(j) ← s_i.
17:            t(rt_j) ← t_j.
18:            rt(t_j) ← rt_j.
19:            t(e_j) ← j and t(e_i) ← j.
20:            rt(i) ← e_j and rt(j) ← e_j.
21:            if e_i = e_j then
22:                e(i) ← rt_j and e_i ← rt_j
23:            end if
24:            t(e_j) ← i.
25:            rt(j) ← e_i and e(j) ← e_i.
26:        end for
27:    end if
28: end procedure
```

Algorithm 6 XTI method for reattaching the x_2 subtree onto x_1.

```
1: procedure REATTACH SUBTREE(rt, t, e, p, s, entering arc, x_2, y_2)
2:     Let t_{y1} ← t(y_1)
3:     t(e(y_2)) ← t(y_1)
4:     rt(t(y_1)) ← e(y_2)
5:     t(y_1) ← y_2
6:     rt(y_2) ← y_1
7:     p(y_2) ← y_1
8:     if y_1 equals the "to" node in the entering arc then
9:         d(y_2) ← True
10:    else
11:        d(y_2) ← False
12:    end if
13:    Let x* = p(t(y_1))
14:    if x* = 0 then
15:        x* ← x_1
16:        for Every node i on the path from y_1 to x* do
17:            if i ≠ x* then
18:                e(i) ← e(y_2)
19:            end if
20:            s(i) ← s(i) + s(y_2)
21:        end for
22:    else
23:        for Every node i on the path from y_1 to x* do
24:            e(i) ← e(y_2)
25:            s(i) ← s(i) + s(y_2)
26:        end for
27:    end if
28: end procedure
```

2.5 Summary of the Single-Pivot Network Simplex Algorithm

Algorithm 7 presents all the steps of the single-pivot network simplex method considering all implementation details discussed in previous sections. The next section presents an iterative example of the single-pivoting method to solve minimum-cost network flow problems.

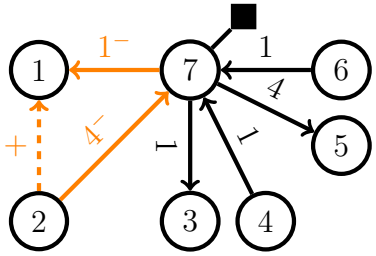
Algorithm 7 The Single-Pivot Network Simplex Method.

1: **procedure** Single-Pivot Network Simplex Method(V,E)
2: Let $M \leftarrow \sum_{(i,j)\in A} |c_{ij}|$ or 1 if $M = 0$
3: Let $n \leftarrow |N| + 1$
4: Let $p \leftarrow [\underbrace{n, n, n, \ldots, n, n, n}_{\text{length } n-1}, 0]$
5: Let $s \leftarrow [\underbrace{1, 1, 1, \ldots, 1, 1, 1}_{\text{length } n-1}, n]$
6: Let $e \leftarrow [1, 2, 3, 4, \ldots, n-3, n-2, n-1]$
7: Let $t \leftarrow [2, 3, 4, \ldots, n-1, n, n+1, 1]$
8: Let $rt \leftarrow [1, 2, 3, 4, \ldots, n-3, n-2, n-1]$
9: Let $w \leftarrow [\underbrace{0, 0, 0, \ldots, 0, 0, 0}_{\text{length } n}]$
10: Let $d \leftarrow [\underbrace{False, False, False, \ldots, False, False, False}_{\text{length } n-1}, 0]$
11: **for** all $i \in N$ **do**
12: $w_i \leftarrow M$ if $b_i > 0$
13: $d_i \leftarrow True$ if $b_i > 0$
14: **end for**
15: Calculate the reduced costs c^{π}_{ij} for block pivoting
16: **while** there exists an improving c^{π}_{ij} via block pivoting **do**
17: Let the entering arc equal the most improving c^{π}_{ij}
18: Call **Algorithm 1**
19: Find minimum arc and then update flows
20: Call **Algorithm 2, 3, 4, 5, and 6**
21: Recalculate the reduced costs c^{π}_{ij} for block pivoting
22: **end while**
23: **Return** $z^* = \sum_{(i,j)\in A} c_{ij}x_{ij}$ and the optimal spanning tree basis.
24: **end procedure**

2.6 Iterative Example of the Single-Pivot Network Simplex Method

This section solves **Example** 1 using the single-pivot network simplex. Remind that Figure 2.1 shows the network and Figure 2.2 presents the initial spanning tree. For this example, improving arcs are not selected on the traditional best-arc approach. Instead, the block pivoting rule is used. The size of each block is three and they are shown using dashed lines in the tables. In addition, $M = 46$ is used to compute the cost of the artificial arcs.

Iteration One, $z = 276$



n	t	rt	p	s	e	d	w
1	2	7	7	1	1	F	0
2	3	1	7	1	2	T	46
3	4	2	7	1	3	F	0
4	5	3	7	1	4	T	46
5	6	4	7	1	5	F	0
6	7	5	7	1	6	T	46
7	1	6	0	7	6	0	0

(i,j)	$w_i - w_j - c_{ij}$	c^{π}_{ij}	***Status***
(1, 3)	0 - 0 - 7	7	Lower
(1, 4)	0 - 46 - 5	-51	Lower
P^*(2, 1)	46 - 0 - 10	36	Lower
(2, 5)	46 - 0 - 6	40	Lower
(3, 5)	0 - 0 - 6	-6	Lower
(4, 6)	46 - 46 - 8	-8	Lower
(5, 6)	0 - 46 - 1	-47	Lower
(6, 3)	46 - 0 - 3	43	Lower

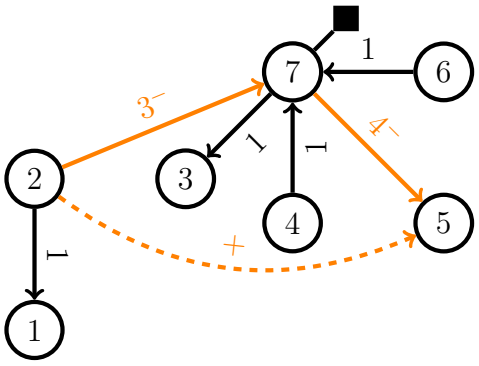
$$\Delta f = \min\{u_{21}, x_{71}\} = \min\{5, 1, 4\} = 1$$

Figure 2.9: Iteration one of **Example** 1.

Firstly, the node potentials w and reduced costs c^{π}_{ij} are computed for each node and arc, respectively, as shown in Figure 2.9. Since arc (2,1) has the largest reduced cost

among the arcs within the first block (arcs (1,3), (1,4), and (2,1)), then it is selected as the entering arc forming a cycle in the spanning tree basis. The flow in the cycle is updated such that the new spanning tree has $x_{21} = 1, x_{27} = 3$, and $x_{71} = 0$. Consequently, arc (7,1) is removed from the spanning tree.

Iteration Two, $z = 240$



n	t	rt	p	s	e	d	w
1	3	2	2	1	1	F	36
2	1	7	7	2	1	T	46
3	4	1	7	1	3	F	0
4	5	3	7	1	4	T	46
5	6	4	7	1	5	F	0
6	7	5	7	1	6	T	46
7	2	6	0	7	6	0	0

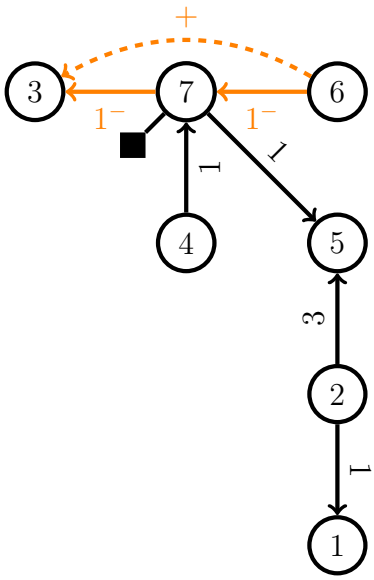
$\Delta f = \min\{u_{25}, x_{75}, x_{27}\} = 3$

(i,j)	$w_i - w_j - c_{ij}$	c_{ij}^{π}	***Status***
(1, 3)	36 - 0 - 7	29	Lower
(1, 4)	36 - 46 - 5	-15	Lower
(2, 1)	46 - 36 - 10	0	BV
P^*(2, 5)	46 - 0 - 6	40	Lower
(3, 5)	0 - 0 - 6	-6	Lower
(4, 6)	46 - 46 - 8	-8	Lower
(5, 6)	0 - 46 - 1	-47	Lower
(6, 3)	46 - 0 - 3	43	Lower

Figure 2.10: Iteration two of **Example** 1.

On the second iteration (**Figure** 2.10), the dual variables and reduced costs are calculated for all nodes and arcs, respectively. In this case, the arc with the largest reduced cost in the second block is (2,3), which results in this being the entering arc. When this arc is introduced, a cycle between arcs (2,5), (7,5), and (2,7) is created. The change of flow, in this case, is $\Delta_f = 3$, and the leaving arc is (2,7). The flow is updated so that $x_{25} = 3, x_{75} = 1$, and $x_{27} = 0$ in the next spanning tree basis. Therefore, arc (2,7) is removed.

Iteration Three, $z = 120$



n	t	rt	p	s	e	d	w
1	6	2	2	1	1	F	-4
2	1	5	5	2	1	T	6
3	4	7	7	1	3	F	0
4	5	3	7	1	4	T	46
5	2	4	7	3	1	F	0
6	7	1	7	1	6	T	46
7	1	6	0	7	6	0	0

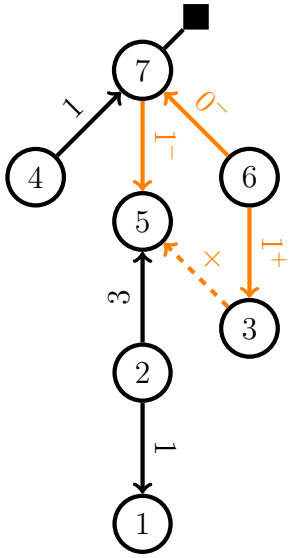
$\Delta f = \min\{u_{63}, x_{73}, x_{67}\} = 1$

(i,j)	$w_i - w_j - c_{ij}$	c^{π}_{ij}	***Status***
(1, 3)	-4 - 0 - 7	-11	Lower
(1, 4)	-4 - 46 - 5	-55	Lower
(2, 1)	6 - -4 - 10	0	BV
(2, 5)	6 - 0 - 6	0	BV
(3, 5)	0 - 0 - 6	-6	Lower
(4, 6)	46 - 46 - 8	-8	Lower
(5, 6)	0 - 46 - 1	-47	Lower
P^*(6, 3)	46 - 0 - 3	43	Lower

Figure 2.11: Iteration three of **Example** 1.

The same process is repeated for the next iterations. The third block is used for the third iteration (**Figure** 2.11) and arcs (6,3) and (7,3) are the entering and leaving arcs, respectively. For the fourth iteration (**Figure** 2.12), the second block is used. This is because there are no improving arcs in the first block. In this case, arc (3,5) is the entering one while arc (7,6) is the leaving one. Furthermore, notice that the change of flow in the fourth iteration is determined by the minimum between u_{35}, x_{75}, x_{76}, and $u_{63} - x_{63}$.

Iteration Four, $z = 77$



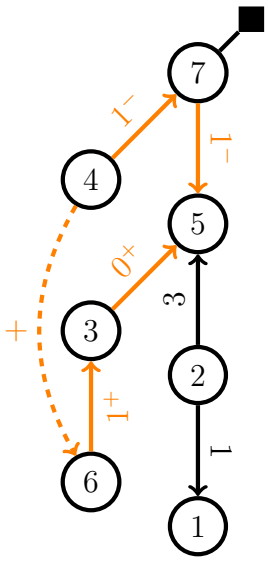
n	t	rt	p	s	e	d	w
1	3	2	2	1	1	F	-4
2	1	7	7	2	1	T	6
3	4	1	7	1	3	F	43
4	5	3	7	1	4	T	46
5	6	4	7	1	5	F	0
6	7	5	7	1	6	T	46
7	2	6	0	7	6	0	0

$$\Delta f = \min\{u_{35}, x_{75}, x_{76}, u_{63} - x_{63}\} = 0$$

(i,j)	$w_i - w_j - c_{ij}$	c^{π}_{ij}	***Status***
(1, 3)	-4 - 0 - 7	-54	Lower
(1, 4)	-4 - 46 - 5	-55	Lower
(2, 1)	6 - -4 - 10	0	BV
(2, 5)	6 - 0 - 6	0	BV
P^*(3, 5)	43 - 0 - 6	37	Lower
(4, 6)	46 - 46 - 8	-8	Lower
(5, 6)	0 - 46 - 1	-47	Lower
(6, 3)	46 - 43 - 3	0	BV

Figure 2.12: Iteration four of **Example** 1.

Iteration Five, $z = 77$



n	t	rt	p	s	e	d	w
1	7	2	2	1	1	F	-4
2	1	6	5	2	1	T	6
3	6	5	5	2	6	T	6
4	5	7	7	1	4	T	46
5	3	4	7	5	1	F	0
6	2	3	3	1	6	T	9
7	4	1	0	7	1	0	0

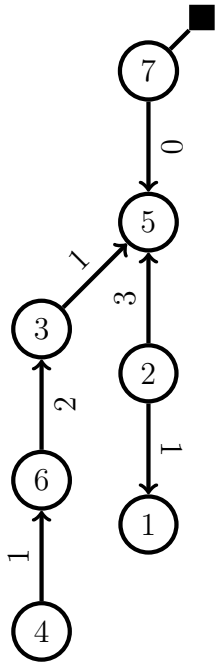
$$\Delta f = \min\{u_{46}, u_{63} - x_{63}, u_{35} - x_{35}, x_{75}, x_{47}\} = 1$$

(i,j)	$w_i - w_j - c_{ij}$	c_{ij}^{π}	***Status***
(1, 3)	-4 - 6 - 7	-17	Lower
(1, 4)	-4 - 46 - 5	-55	Lower
(2, 1)	6 - -4 - 10	0	BV
(2, 5)	6 - 0 - 6	0	BV
(3, 5)	6 - 0 - 6	0	BV
$P*$(4, 6)	46 - 9 - 8	29	Lower
(5, 6)	0 - 9 - 1	-10	Lower
(6, 3)	9 - 6 - 3	0	BV

Figure 2.13: Iteration five of **Example** 1.

During iteration five (**Figure** 2.13), the entering arc is determined from the second block. This makes arc (4,6) the entering one, thus creating a cycle between arcs (4,6), (6,3), (3,5), (7,5), and (4,7). In this case, the minimum occurs at arc (7,4), which makes it the leaving arc. After the node potentials and reduced costs are updated as shown in **Figure** 2.14, one can see that no improving arcs exist. Therefore, the spanning tree basis presented in **Figure** 2.14 is optimal and the corresponding optimal objective function value is $z^* = 48$.

Final Results, $z^* = 48$



n	t	rt	p	s	e	d	w
1	7	2	2	1	1	F	-4
2	1	4	5	2	1	T	6
3	6	5	5	3	4	T	6
4	2	6	6	1	4	T	17
5	3	7	7	6	1	F	0
6	4	3	3	2	4	T	9
7	5	1	0	7	1	0	0

(i,j)	$w_i - w_j - c_{ij}$	c_{ij}^{π}	***Status***
(1, 3)	-4 - 6 - 7	-17	Lower
(1, 4)	-4 - 17 - 5	-26	Lower
(2, 1)	6 - -4 - 10	0	BV
(2, 5)	6 - 0 - 6	0	BV
(3, 5)	6 - 0 - 6	0	BV
(4, 6)	17 - 9 - 8	0	BV
(5, 6)	0 - 9 - 1	-10	Lower
(6, 3)	9 - 6 - 3	0	BV

Figure 2.14: Final result of **Example** 1.

3 The Double-Pivot Network Simplex Method

This chapter presents the major theoretical and algorithmic developments of the double-pivot network simplex method. It discusses the challenges of introducing two cycles into a spanning tree basis, how to formulate and solve the change of flow problem, the various pivoting types that can occur, the labeling procedure for double-pivoting, the complete double-pivot network simplex algorithm, and a numerical example.

3.1 Multi-Cycling

The primary difference between the single and double-pivot network simplex methods is the number of arcs entering and leaving the spanning tree basis per iteration. As previously mentioned, the single-pivoting method introduces and removes one arc per iteration, while the double-pivoting method introduces and removes two arcs per iteration if possible - there are cases where only one arc is introduced and removed at a time. Introducing two arcs into a spanning tree basis induces two cycles within the tree. The overlapping aspect of these cycles provokes various issues that are explored with casework using the network presented in **Figure** 3.1 where $(7, 11)$ and $(10, 13)$ are the entering arcs.

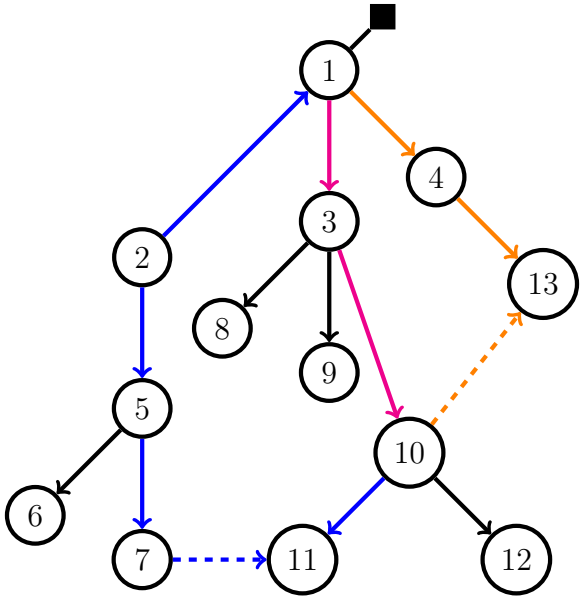


Figure 3.1: Spanning tree basis used for the overlapping casework with cycle one denoted in blue, cycle two in orange, and overlapping arcs in magenta.

When two arcs enter the spanning tree basis, there are four possible cases for cycles: only one overlapping arc leaves the basis (Section 3.1.1), two different overlapping arcs leave the basis (Section 3.1.2), one overlapping and one non-overlapping arcs leave the basis (Section 3.1.3), and only non-overlapping arcs leave the basis (Section 3.1.4).

3.1.1 Case One: Only One Overlapping Arc Leaves the Basis

Let $(3, 10)$ be the leaving arc for the blue and orange cycles. The resulting graph is presented in **Figure** 3.2. Notice that a cycle still exists; therefore, it is no longer a tree. Consequently, only one overlapping arc cannot leave the basis.

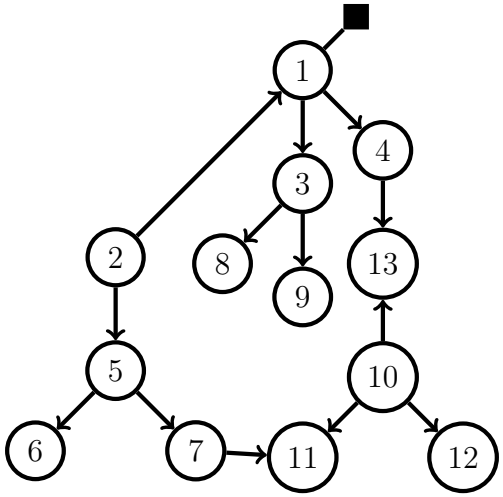


Figure 3.2: Spanning tree "basis" in which both cycles converge on $(3, 10)$ being the leaving arc.

3.1.2 Case Two: Two Different Overlapping Arcs Leave the Basis

Let $(1, 3)$ and $(3, 10)$ be the leaving arcs for the blue and orange cycles. **Figure** 3.3 demonstrates resulting graph. Not only does a cycle exist, but it is also disconnected making it no longer a tree. Therefore, two overlapping arcs cannot leave the basis.

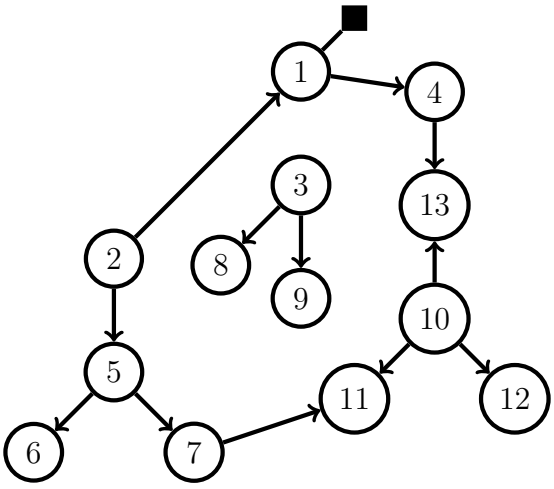


Figure 3.3: Spanning tree "basis" in which both cycles converge on $(1, 3)$ and $(3, 10)$ being the leaving arcs.

3.1.3 Case Three: One Overlapping and One Non-overlapping Arc Leave the Basis

Let $(2, 5)$ be the leaving arc for the blue cycle while $(3, 10)$ is the leaving arc for the orange cycle. **Figure** 3.4 shows the spanning tree basis after both arcs are removed. Since there are no cycles or disjoint nodes in the tree, then one overlapping and one non-overlapping arc can leave the basis.

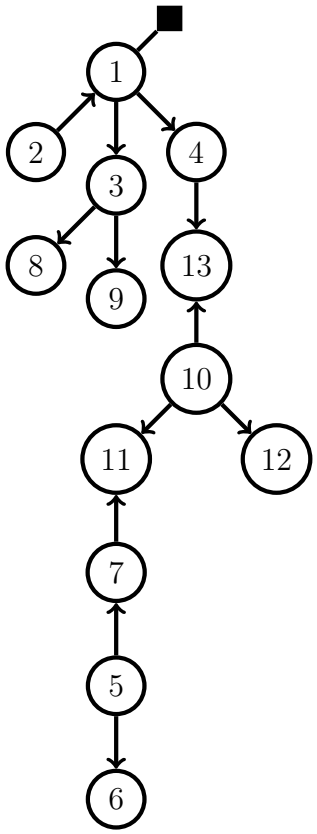


Figure 3.4: Spanning tree basis in which $(2, 5)$ and $(3, 10)$ are the leaving arcs.

3.1.4 Case Four: Only Non-overlapping Arcs Leave the Basis

Let $(2, 5)$ be the leaving arc for the blue cycle, and the $(1, 4)$ be the leaving arc for the orange cycle. **Figure** 3.5 shows the spanning tree basis for this case. One can see that no cycles or disjoint nodes exist in the tree. Therefore, only non-overlapping arcs can leave the basis.

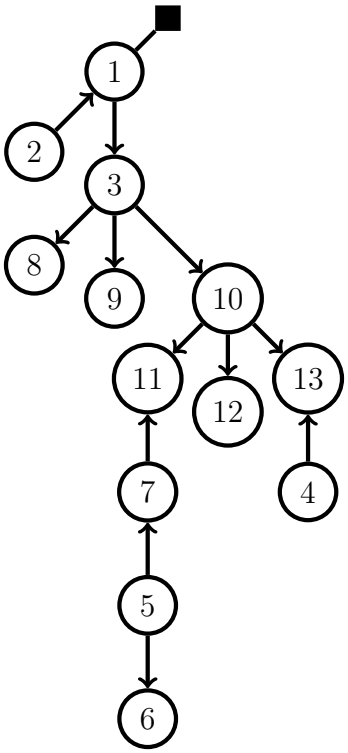


Figure 3.5: Spanning tree basis in which $(2, 5)$ and $(1, 4)$ are the leaving arcs.

3.2 Two-Variable Change of Flow Problem

One can see that cases one (Section 3.1.1) and two (Section 3.1.2) are not valid pivots while cases three (Section 3.1.3) and four (Section 3.1.4) are. If entering two arcs results in case one, one can choose the overlapping arc and the minimum non-overlapping arc from both cycles or choose one minimum non-overlapping arc from each cycle to leave the basis. If two entering arcs result in case two, one can choose the minimum overlapping arc and the minimum non-overlapping arc from both cycles or choose one minimum non-overlapping arc from each cycle to leave the basis. If entering two arcs results in cases three and four, then one can find the minimum arcs in cycles one and two, respectively, and alter the amount of flow on the cycle in the same manner as the single-pivoting method.

Observe that determining which arcs to leave the basis is defined by the change of flow. Therefore, consider **Example** 2 to demonstrate the change of flow when two arcs enter the basis and there are overlapping arcs between both cycles.

Example 2 *Consider the spanning tree rooted at node 1 presented in* ***Figure*** *3.6.*

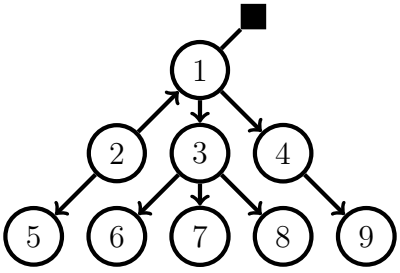


Figure 3.6: Spanning tree of **Example** 2.

Let x_1 and x_2 be variables denoting the change of flow in cycles one and two, respectively. Let b_i denote the arc values found in the minimum overlapping and non-overlapping arcs. Two types of overlapping cases happen independently of each other: the positive-flow overlapping case ($x_1 + x_2 \leq b_1$) and the mixed flow overlapping case ($-x_1 + x_2 \leq b_1$ and $x_1 - x_2 \leq b_2$).

3.2.1 Change of Flow for the Positive Flow Case

The positive flow overlapping case is where the flow of overlapping arcs between both cycles occurs in the same direction, as shown in **Figure** 3.7 where the overlapping arc, colored in magenta, is experiencing a double decreasing flow from both cycles one (blue) and two (orange).

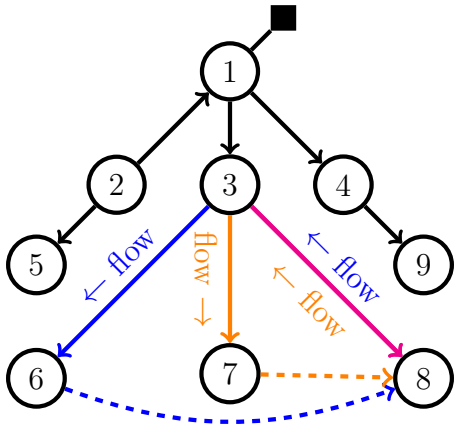


Figure 3.7: Example of cycle one (blue) and cycle two (orange) in the same direction where the overlapping arc is colored in magenta.

There are only two possible outcomes for the positive flow case for the overlapping arc(s) depending on how the cycles are oriented: $x_1 + x_2 \leq u_{ij} - x_{ij}$ (increasing flow) or $x_1 + x_2 \leq x_{ij}$ (decreasing flow). Note that each arc is bounded between $0 \leq x_{ij} \leq u_{ij}$

for all $(i, j) \in A$ such that the lowest possible change of flow value for any of these cycles is zero. The change of flow for the positive flow case can be modeled as the following two-variable linear program:

$$\begin{array}{lll} \max \quad z = & c_1x_1 + c_2x_2 & \\ \text{Subject to :} & x_1 + x_2 \leq b_1 & \xrightarrow{\text{Standard Form}} \\ & 0 \leq x_1 \leq b_2 & \\ & 0 \leq x_2 \leq b_3 & \end{array} \quad \begin{array}{lll} \max \quad z = & c_1x_1 + c_2x_2 \\ \text{Subject to :} & x_1 + x_2 + s_1 = b_1 \\ & x_1 + s_2 = b_2 \\ & x_2 + s_3 = b_3 \end{array} \tag{3.1}$$

where c_1 and c_2 denote the reduced costs, c_{ij}^{π}, of the entering arcs corresponding to each cycle, b_1 represents the minimum overlapping arc, b_2 and b_3 denote the minimum change of flow for the arcs on cycles one and two that are non-overlapping, and s_1, s_2 and s_3 are slack variables used to convert the two-variable problem to standard form.

3.2.2 Change of Flow for the Mixed Flow Case

The mixed flow overlapping case is where the flow of overlapping arcs between both cycles occurs in opposite directions, as presented in **Figure** 3.8.

The change of flow for the overlapping arc(s) is determined by either $-x_1 + x_2$ or $x_1 - x_2$. Notice that in this case, flow can be increased by the amount $u_{ij} - x_{ij}$ or decreased by the amount x_{ij}. In other words, both $-x_1 + x_2 \leq u_{ij} - x_{ij}$ and $x_1 - x_2 \leq x_{ij}$ need to be considered since flow is occurring in opposite directions. The

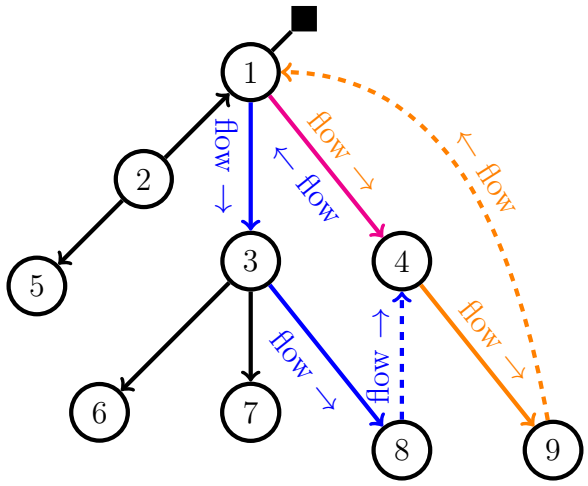


Figure 3.8: Example of cycle one (blue) and cycle two (orange) in the opposite direction where the overlapping arc is colored in magenta.

mixed flow case can be modeled as the following two-variable linear program:

$$\begin{array}{lll} \max & z = & c_1x_1 + c_2x_2 \\ \text{Subject to :} & & -x_1 + x_2 \leq b_1 \\ & & x_1 - x_2 \leq b_2 \\ & & 0 \leq x_1 \leq b_3 \\ & & 0 \leq x_2 \leq b_4 \end{array} \xrightarrow{\text{Standard Form}} \begin{array}{lll} \max & z = & c_1x_1 + c_2x_2 \\ \text{Subject to :} & & -x_1 + x_2 + s_1 = b_1 \\ & & x_1 - x_2 + s_2 = b_2 \\ & & x_1 + s_3 = b_3 \\ & & x_2 + s_4 = b_4 \end{array} \tag{3.2}$$

where b_1 and b_2 represent the amount by which flow can be increased or decreased, respectively, b_3 and b_4 denote the minimum change of flow for the arcs on cycles one and two that are independent of the overlapping arcs, and s_1, s_2, s_3 and s_4 are slack variables used to convert the two-variable problem to standard form. The feasible region of this linear program is presented in **Figure** 3.9 with the $x_1 - x_2$ constraint in blue, the $-x_1 + x_2$ constraint in orange, the $x_1 \leq x_{ij}$ (or $x_1 \leq u_{ij} - x_{ij}$) constraint in purple, the $x_2 \leq x_{ij}$ (or $x_2 \leq u_{ij} - x_{ij}$) constraint in green, feasible region in gray.

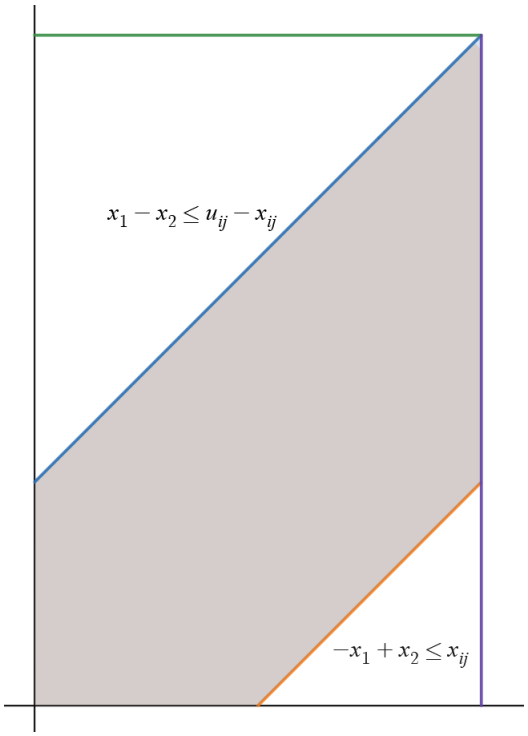


Figure 3.9: Feasible region of the two-variable linear program from the mixed flow case. Observe that the right-hand side values on the constraints can be swapped depending on the order in which minimum arcs and cycles are labeled and computed.

There is a need to produce a computationally efficient approach to solving these two-variable problems. The following section discusses this.

3.3 Solving the Two-Variable Linear Program

Observe that the change of flow in the single-pivoting method is determined by solving a one-dimensional search problem. In contrast, the double-pivoting method determines the change of flow by solving a two-dimensional search problem. Solving a two-dimensional problem is computationally more difficult than solving a one-dimensional problem. Therefore, to make the double-pivoting method more effective, one needs to solve these two-variable linear programs using a faster technique than traditional

methods such as the simplex method or interior point methods[3]. This is because the first requires updating a basis inverse at each iteration while the second requires solving a system of equations using either the LU or Cholesky decomposition methods[3]. Additionally, if a traditional linear programming algorithm is used to solve the two-variable problems, then the double-pivot simplex method cannot be categorized as a multidimensional search method according to **Definition** 1 since neither the simplex method nor interior point algorithms are techniques designed to perform multidimensional searches[32].

Definition 1 *To be characterized as a multidimensional search method, the algorithm must:*

1. *Search over multiple search directions at each iteration (more than one).*
2. *Optimally solve a multidimensional subspace problem to determine the step length for each search direction using methods designed to perform multidimensional searches.*

Shamos and Hoey created one of the first techniques to perform multidimensional searches. Their method solves two-variable linear programs by determining if any two constraints have a common intersection[29]. Afterward, Megiddo proposed algorithms to solve two and three-variable LPs[25]. Dyer independently derived a similar approach to Megiddo to solve two and three-variable linear programs[13]. Vitor and Easton developed a method to find an optimal basis for two-variable linear programs called the slope algorithm, which handles degenerate two-variable linear programs as well[33]. A dual space method for solving two-variable linear programs was created by Vitor in

2018 in which he developed the ratio algorithm for two-constraint linear programs[31]. Additionally, Jamrunroj and Boonperm propose a method to solve two-variable linear programs by considering only the coefficient of the constraints[18]. In terms of non-linear optimization, Santos et al. created an algorithm to solve three-variable non-linear polynomial optimization problems[28].

While these methods to perform multidimensional searches are fast and could be used, a faster approach to solve two-variable linear programs is created for this thesis. This is because the two-variable problems for the double-pivoting method are much simpler. Notice that **Models** (3.1) and (3.2) have at most four constraints and the constraint matrix is composed of either 0, +1, or -1 values. An optimal solution to the two-variable linear program, which determines the leaving arcs, occurs when the arc's constraint is binding at an extreme point (when the slack variable, s_i, is equal to zero). Given the special structure of these two-variable problems, it is easy to enumerate all extreme points. The following sections discuss this procedure for the positive flow and mixed flow model cases. The casework for which non-overlapping arcs leave the basis is not considered, since this is the same as performing a one-dimensional search. That is, computing the change of flow in the same manner as the single-pivot network simplex method.

3.3.1 Positive Flow Model Cases

This section investigates every case where an arc's constraint is binding at an extreme point for **Model** (3.1). The section starts with the case of cycle one minimum arc and minimum overlapping arc leaving the basis.

Case One: Cycle One Minimum Arc and the Minimum Overlapping Arc Leave the Basis

The minimum arc on cycle one and the overlapping arc $x_1 + x_2 \leq b_1$ leave the basis with $s_1 = 0$ and $s_2 = 0$. Hence,

$$\begin{array}{l} z = c_1 b_2 + c_2(b_3 - s_3) \\ \\ b_3 - s_3 = b_1 - b_2 \end{array} \quad \longrightarrow \quad z = c_1 b_2 + c_2(b_1 - b_2).$$

Consequently, $x_1 = b_2$ and $x_2 = b_1 - b_2$. Substituting these back into **Model** (3.1) results in $z = b_2(c_1 - c_2) + c_2 b_1$ as long as $b_1 \geq b_2$ and $b_1 - b_2 \leq b_3$ are true.

Case Two: Cycle One Minimum Arc and Cycle Two Minimum Arc Leave the Basis

The minimum arc on cycle one and the minimum arc on cycle two leave the basis with $s_2 = 0$ and $s_3 = 0$. Therefore,

$$z = c_1 b_2 + c_2 b_3$$

$$b_2 + b_3 - b_1 = -s_1.$$

In this case, $x_1 = b_2$ and $x_2 = b_3$. Substituting these back into **Model** (3.1) results in $z = c_1 b_2 + c_2 b_3$ as long as $b_2 + b_3 \leq b_1$ is true.

Case Three: Cycle Two Minimum Arc and the Minimum Overlapping Arc Leave the Basis

The minimum arc on cycle one and the minimum overlapping arc leaves the basis with $s_3 = 0$ and $s_1 = 0$. Notice that

$$\begin{array}{c} z = c_1(b_2 - s_2) + c_2 b_3 \\ b_2 - s_2 = b_1 - b_3 \end{array} \longrightarrow \quad z = c_1(b_1 - b_3) + c_2 b_3.$$

Thus, $x_1 = b_1 - b_3$ and $x_2 = b_3$. Substituting these back into **Model** (3.1) produces $z = c_1(b_1 - b_3) + c_2 b_3$ as long as $b_1 \geq b_3$ and $b_1 - b_3 \leq b_2$ are true.

Case Four: Only the Minimum Overlapping Arc Leaves the Basis

The minimum overlapping arc is the only one that leaves the basis with $s_1 = 0$. Thus,

$$z = c_1(b_2 - s_2) + c_2(b_3 - s_3)$$

$$b_2 + b_3 - b_1 = s_2 + s_3.$$

However, taking advantage of the fact that if the minimum overlapping arc is the only arc to leave, then either $x_1 = 0$ or $x_2 = 0$. If $x_1 = 0$, then

$$\begin{array}{l} z = c_2(b_3 - s_3) \\ \\ b_3 - s_3 = b_1 \end{array} \quad \longrightarrow \quad z = c_2 b_1.$$

Consequently, $x_1 = 0$ and $x_2 = b_1$. Substituting these back into **Model** (3.1) results in $z = c_2 b_1$ as long as $b_1 \leq b_2$ is true.

If $x_2 = 0$, then

$$\begin{array}{l} z = c_1(b_2 - s_2) \\ \\ b_2 - s_2 = b_1 \end{array} \quad \longrightarrow \quad z = c_1 b_1.$$

In this case, $x_1 = b_1$ and $x_2 = 0$. Substituting these back into **Model** (3.1) results in $z = c_1 b_1$ as long as $b_1 \leq b_3$ is true.

3.3.2 Positive Flow Model Procedure

For the positive flow model, one needs to find the minimum overlapping arc in cycles one and two, and the minimum non-overlapping arcs in cycles one and two, and set their values to b_1, b_2, and b_3, respectively. Let c_1 and c_2 denote the reduced costs, c_{ij}^{π}, of the entering arcs of cycles one and two, respectively. **Algorithm** 8 formalizes the procedure discussed in **Section** 3.3.1.

Algorithm 8 Positive Flow Model Procedure

procedure POSITIVE FLOW PROCEDURE(b_1, b_2, b_3, c_1, c_2)
    Let $s_1 \leftarrow b_2(c_1 - c_2) + c_2 b_1$, if $b_1 - b_2 \leq b_3$ and $b_1 \geq b_2$, else $-\infty$
    Let $s_2 \leftarrow c_1 b_2 + c_2 b_3$, if $b_2 + b_3 \leq b_1$, else $-\infty$
    Let $s_3 \leftarrow b_2(c_2 - c_1) + c_1 b_1$, if $b_1 - b_3 \leq b_2$ and $b_1 \geq b_3$, else $-\infty$
    Let $s_4 \leftarrow c_1 b_1$, if $b_1 \leq b_2$, else $-\infty$
    Let $s_5 \leftarrow b_1 c_2$, if $b_1 \leq b_3$, else $-\infty$
    Let $i \leftarrow \arg\max(s_1, s_2, s_3, s_4, s_5)$
    if $i = 1$ **then**
        $x_1 \leftarrow b_2$
        $x_2 \leftarrow b_1 - b_2$
        Remove the minimum arc on cycle one and the minimum overlapping arc
    else if $i = 2$ **then**
        $x_1 \leftarrow b_2$
        $x_2 \leftarrow b_3$
        Remove the minimum arc on cycle one and the minimum arc on cycle two
    else if $i = 3$ **then**
        $x_1 \leftarrow b_1 - b_3$
        $x_2 \leftarrow b_3$
        Remove the minimum arc on cycle two and the minimum overlapping arc
    else if $i = 4$ **then**
        $x_1 \leftarrow b_1$
        $x_2 \leftarrow 0$
        Remove the minimum overlapping arc
    else if $i = 5$ **then**
        $x_1 \leftarrow 0$
        $x_2 \leftarrow b_1$
        Remove the minimum overlapping arc
    end if
end procedure

Notice that an arc always leaves the basis since arcs cannot have negative flow due to $b_1, b_2, b_3 \geq 0$. Furthermore, using the arg max function prioritizes the cases where two arcs leave the spanning tree basis in the event of a tie since the function evaluates on a left-to-right occurrence. **Figure** 3.10 presents a graphical depiction of the feasible region and extreme points of the positive flow case.

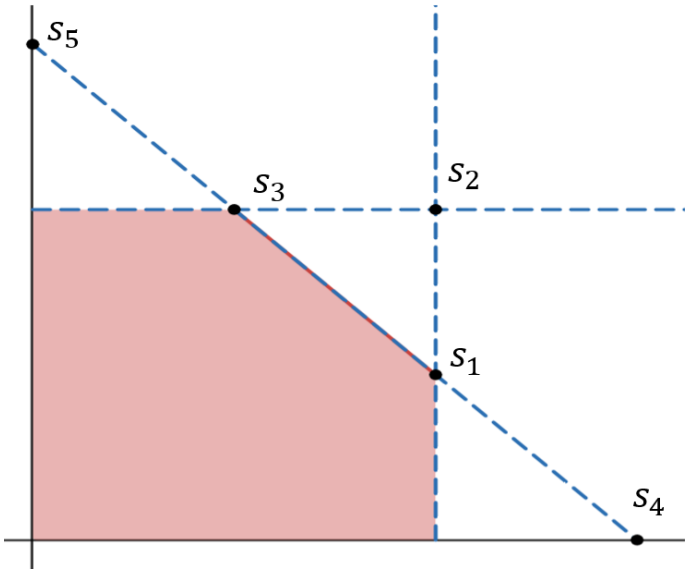


Figure 3.10: Feasible region and extreme points of the two-variable linear program from the positive flow case.

3.3.3 Mixed Flow Model Cases

Similarly to the positive flow model, there is a need to develop casework for the mixed flow model presented in **Model** (3.2). The following sections present the casework.

Case One: Cycle One Minimum Arc and the Minimum Overlapping Arc Leave the Basis

The minimum arc on cycle one and the minimum overlapping arc leaves the basis with $s_1 = 0$ and $s_3 = 0$. This results in

$$\begin{aligned} z &= c_1 b_3 + c_2(b_4 - s_4) \\ b_4 - s_4 &= b_1 + b_3 \\ b_3 - b_4 + s_4 + s_2 &= b_2 \end{aligned} \quad \longrightarrow \quad \begin{aligned} z &= c_1 b_3 + c_2(b_1 + b_3) \\ b_1 + b_2 &= s_2. \end{aligned}$$

Consequently, $x_1 = b_3$ and $x_2 = b_1 + b_3$. Substituting these back into **Model** (3.2) results in $z = c_1 b_3 + c_2(b_1 + b_3)$ as long as $-b_1 \leq b_2$, $-b_1 \leq b_3$, and $b_1 + b_3 \leq b_4$ are true. However, since it is already known that $b_1, b_3, b_2 \geq 0$, then $b_1 + b_3 \leq b_4$ is the only condition that needs to be checked.

Case Two: Cycle Two Minimum Arc and the Minimum Overlapping Arc Leave the Basis

The minimum arc on cycle two and the minimum overlapping arc leaves the basis with $s_1 = 0$ and $s_4 = 0$. Therefore,

$$\begin{array}{l} z = c_1(b_3 - s_3) + c_2 b_4 \\ b_3 - s_3 - b_4 = -b_1 \\ b_3 - s_3 - b_4 + s_2 = b_2 \end{array} \longrightarrow \begin{array}{c} z = c_1(b_4 - b_1) + c_2 b_4 \\ s_2 = b_2 + b_1. \end{array}$$

Thus, $x_1 = b_4 - b_1$ and $x_2 = b_4$. Substituting these back into **Model** (3.2) returns $z = c_1(b_4 - b_1) + c_2 b_4$ as long as $-b_1 \leq b_2$, $b_1 \leq b_4$, and $b_4 - b_1 \leq b_3$ are true. However, since $b_1, b_3, b_2 \geq 0$, then only $b_1 \leq b_4$ and $b_4 - b_1 \leq b_3$ needs to be checked.

Case Three: Cycle One Minimum Arc and the Minimum Overlapping Arc Leave the Basis

The minimum arc on cycle one and the minimum overlapping arc leaves the basis with $s_2 = 0$ and $s_3 = 0$. This results in

$$\begin{array}{c} z = c_1 b_3 + c_2(b_4 - s_4) \\ -b_3 + b_4 - s_4 + s_1 = b_1 \\ -b_3 + b_4 - s_4 = -b_2 \end{array} \longrightarrow \begin{array}{c} z = c_1 b_3 + c_2(b_3 - b_2) \\ s_1 = b_1 + b_2. \end{array}$$

Thus, $x_1 = b_3$ and $x_2 = b_3 - b_2$. Substituting these back into **Model** (3.2) results in $z = c_1 b_3 + c_2(b_3 - b_2)$ as long as $-b_2 \leq b_1$, $b_3 - b_2 \leq b_4$, and $b_2 \leq b_3$ are true. However, since $b_1, b_2, b_3, b_4 \geq 0$, then only $b_3 - b_2 \leq b_4$ and $b_2 \leq b_3$ need to be checked.

Case Four: Cycle Two Minimum Arc and the Minimum Overlapping Arc Leave the Basis

The minimum arc on cycle two and the minimum overlapping arc leaves the basis with $s_2 = 0$ and $s_4 = 0$. One can see that

$$\begin{array}{c} z = c_1(b_3 - s_3) + c_2 b_4 \\ b_3 - s_3 - b_4 - s_1 = -b_1 \\ b_3 - s_3 - b_4 = b_2 \end{array} \longrightarrow \begin{array}{c} z = c_1(b_2 + b_4) + c_2 b_4 \\ b_2 - s_1 = -b_1. \end{array}$$

Hence, $x_1 = b_2 + b_4$ and $x_2 = b_4$. Substituting these back into **Model** (3.2) results in $z = c_1(b_2 + b_4) + c_2 b_4$ as long as $-b_2 \leq b_1$, $b_2 + b_4 \leq b_3$, and $-b_2 \leq b_4$ are true. But since $b_1, b_2, b_3, b_4 \geq 0$, then only $b_2 + b_4 \leq b_3$ needs to be checked.

Case Five: Cycle One Minimum arc and Cycle Two Minimum Arc Leave the Basis

The minimum arc on cycle one and the minimum arc on cycle two leaves the basis with $s_3 = 0$ and $s_4 = 0$. Observe that

$$\begin{array}{l} z = c_1 b_3 + c_2 b_4 \\ b_3 - b_4 = s_1 - b_1 \\ b_3 - b_4 = b_2 - s_2 \end{array} \quad \longrightarrow \quad \begin{array}{c} z = c_1 b_3 + c_2 b_4 \\ \\ b_1 + b_2 = s_1 + s_2. \end{array}$$

In this case, $x_1 = b_3$ and $x_2 = b_4$. Substituting these back into **Model** (3.2) produces $z = c_1 b_3 + c_2 b_4$ as long as $b_4 \leq b_1 + b_3$ and $b_3 - b_4 \leq b_2$ are true.

Case Six: Only the Minimum Overlapping arc Leaves the Basis

The minimum overlapping arc leaves the basis with $s_1 = 0$. In this case

$$z = c_1(b_3 - s_3) + c_2(b_4 - s_4)$$

$$-b_3 + s_3 + b_4 - s_4 = b_1$$

$$b_3 - s_3 - b_4 + s_4 + s_2 = b_2.$$

However, taking advantage of the fact that if the overlapping arc is the only arc to leave, then either $x_1 = 0$ or $x_2 = 0$. Proceeding with $x_1 = 0$:

$$\begin{array}{c} z = c_2(b_4 - s_4) \\ \\ b_4 - s_4 = b_1 \\ \\ b_4 + s_4 + s_2 = b_2 \end{array} \quad \longrightarrow \quad \begin{array}{c} z = c_2 b_1 \\ \\ s_2 = b_2 - b_1. \end{array}$$

Hence, $x_1 = 0$ and $x_2 = b_1$. Substituting these back into **Model** (3.2) results in $z = c_2 b_1$ as long as $b_1 \leq b_4$ is true. For the case of $x_2 = 0$:

$$\begin{array}{c} z = c_1(b_3 - s_3) \\ \\ b_3 - s_3 = -b_1 \\ \\ b_3 - s_3 + s_2 = b_2 \end{array} \quad \longrightarrow \quad \begin{array}{c} z = -c_1 b_1 \\ \\ s_2 = b_2 + b_1. \end{array}$$

This, however, implies that $x_1 = -b_1$, but since arc flows cannot be negative, in that $b_1 \geq 0$, and since $x_1 \geq 0$, then $x_1 = 0$ making $z = 0$ and a needless case for the algorithm to consider.

Case Seven: Only the Minimum Overlapping Arc Leaves the Basis

The minimum overlapping arc leaves the basis with $s_2 = 0$. Notice that

$$z = c_1(b_3 - s_3) + c_2(b_4 - s_4)$$

$$-b_3 + s_3 + b_4 - s_4 + s_1 = b_1$$

$$b_3 - s_3 - b_4 + s_4 = b_2.$$

But, taking advantage of the fact that if the overlapping arc is the only arc to leave, then either $x_1 = 0$ or $x_2 = 0$. Proceeding with $x_1 = 0$:

$$\begin{array}{c} z = c_2(b_4 - s_4) \\ b_4 - s_4 + s_1 = b_1 \\ b_4 - s_4 = -b_2 \end{array} \longrightarrow \begin{array}{c} z = -c_2 b_2 \\ \\ s_1 = b_1 + b_2. \end{array}$$

This, however, implies that $x_2 = -b_2$, but since arcs cannot have negative flow, in that $b_2 \geq 0$, and since $x_2 \geq 0$, then $x_2 = 0$ making $z = 0$ and another needless case for the algorithm to consider. For $x_2 = 0$,

$$\begin{array}{c} z = c_1(b_3 - s_3) \\ -b_3 + s_3 + s_1 = b_1 \\ b_3 - s_3 = b_2 \end{array} \longrightarrow \begin{array}{c} z = c_1 b_2 \\ \\ s_1 = b_1 + b_2. \end{array}$$

Hence, $x_1 = b_2$ and $x_2 = 0$, and substituting these back into **Model** (3.2) results in $z = c_2 b_2$ as long as $b_2 \leq b_3$ is true.

3.3.4 Mixed Flow Model Procedure

For the mixed flow model, one needs to find the minimum overlapping arc on cycles one and two for both $-x_1 + x_2$ and $x_1 - x_2$, and the minimum non-overlapping arcs on both cycles and have their values be b_1, b_2, b_3 and b_4 respectively. Let c_1 and c_2 denote the reduced costs, c_{ij}^{π}, of the entering arcs of cycles one and two. **Algorithm** 9 presents the procedure and **Figure** 3.11 shows a graphical depiction of the feasible region and extreme points of the mixed flow case.

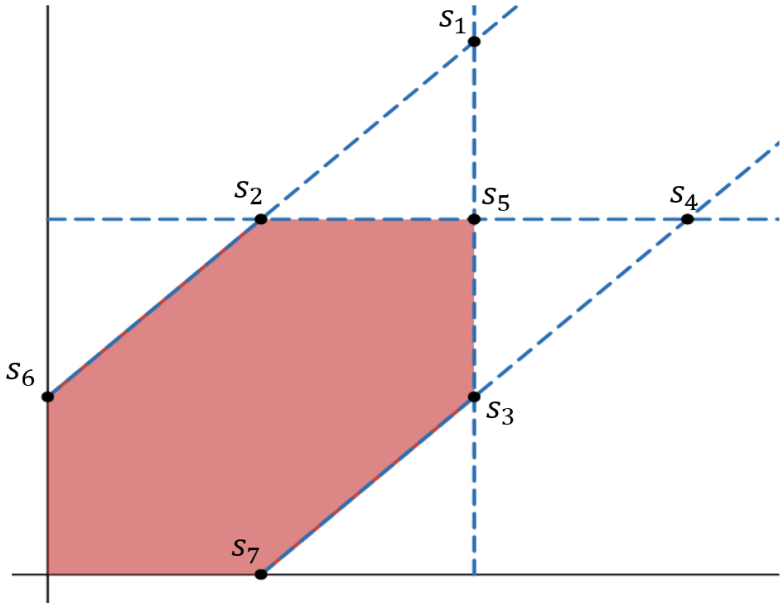


Figure 3.11: Feasible region and extreme points of the two-variable linear program from the mixed flow case.

Similarly, an arc will always leave the basis since $b_1, b_2, b_3, b_4 \geq 0$. Moreover, in the event of a tie, the cases where two arcs leave the spanning tree basis are chosen due to the arg max function.

Algorithm 9 Mixed Flow Model Procedure

procedure MIXED FLOW PROCEDURE($b_1, b_2, b_3, b_4, c_1, c_2$)
    Let $s_1 \leftarrow c_1 b_3 + c_2(b_1 + b_3)$, if $b_1 + b_3 \leq b_4$, else $-\infty$
    Let $s_2 \leftarrow c_1(b_4 - b_1) + c_2 b_4$, if $b_1 \leq b_4$ and $b_4 - b_1 \leq b_3$, else $-\infty$
    Let $s_3 \leftarrow c_1 b_3 + c_2(b_3 - b_2)$, if $b_3 - b_2 \leq b_4$ and $b_2 \leq b_3$, else $-\infty$
    Let $s_4 \leftarrow c_1(b_2 + b_4) + c_2 b_4$, if $b_2 + b_4 \leq b_3$, else $-\infty$
    Let $s_5 \leftarrow c_1 b_3 + c_2 b_4$, if $b_4 \leq b_1 + b_3$ and $b_3 - b_4 \leq b_2$, else $-\infty$
    Let $s_6 \leftarrow c_2 b_1$, if $b_1 \leq b_4$, else $-\infty$
    Let $s_7 \leftarrow b_2 c_2$, if $b_2 \leq b_3$, else $-\infty$
    Let $i \leftarrow \arg\max(s_1, s_2, s_3, s_4, s_5, s_6, s_7)$
    if $i = 1$ **then**
        $x_1 \leftarrow b_3$
        $x_2 \leftarrow b_1 + b_3$
        Remove the minimum arc on cycle one and the overlapping arc
    else if $i = 2$ **then**
        $x_1 \leftarrow b_4 - b_1$
        $x_2 \leftarrow b_4$
        Remove the minimum arc on cycle two and the overlapping arc
    else if $i = 3$ **then**
        $x_1 \leftarrow b_3$
        $x_2 \leftarrow b_3 - b_2$
        Remove the minimum arc on cycle one and the overlapping arc
    else if $i = 4$ **then**
        $x_1 \leftarrow b_2 + b_4$
        $x_2 \leftarrow b_4$
        Remove the minimum arc on cycle two and the overlapping arc
    else if $i = 5$ **then**
        $x_1 \leftarrow b_3$
        $x_2 \leftarrow b_4$
        Remove the minimum arcs on cycle one and cycle two
    else if $i = 6$ **then**
        $x_1 \leftarrow 0$
        $x_2 \leftarrow b_1$
        Remove the minimum overlapping arc
    else if $i = 7$ **then**
        $x_1 \leftarrow b_2$
        $x_2 \leftarrow 0$
        Remove the minimum overlapping arc
    end if
end procedure

3.4 Double-Single Pivots

A particular case exists in the two-cycle problem in which no overlapping arcs interfere with the upper bounds of the other two minimum arcs on both cycles, or no overlapping arcs exist at all. The feasible region of the two-variable linear program for both situations is depicted in **Figure** 3.12.

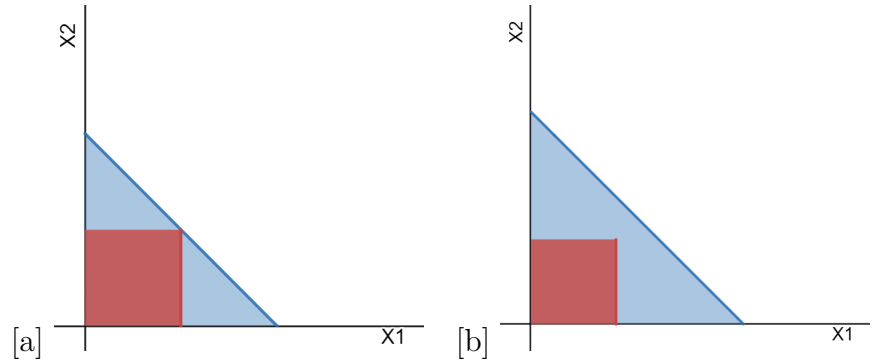


Figure 3.12: Graphical depiction of two cases where double-single pivoting can occur with an overlapping arc.

The feasible region (red) on **Figure** 3.12a denotes the situation in which the overlapping arc (blue line) intersects the two minimum non-overlapping arcs (red square). In this scenario, two arcs can leave the spanning tree basis on the top-right extreme point. However, **Figure** 3.12b can only double-single pivot the minimum non-overlapping arcs in both cycles on that same extreme point. Regarding the non-overlapping arcs situation, one can single-pivot both cycles and avoid the extra computational effort of setting up and solving the two-variable problem.

Additionally, if the minimum arcs of both cycles are found before entering the two-variable problem, both arcs are not identical, and both arcs have their change of flow

equals zero, then both arcs can leave the spanning tree basis with a zero change of flow. In this case, there is no need to set up and solve the two-variable problem.

3.5 False Double-Pivots

False double pivots occur when the overlapping arc is the only arc to leave the basis in the two-variable problem. This refers to cases four and five of the positive flow problem (Section 3.3.2) and cases six and seven of the mixed flow problem (Section 3.3.3). One can view false double-pivots as an unnecessary computational effort since a two-variable problem has to be set up and solved for only one arc to leave the spanning tree basis. However, this is unavoidable as it is difficult to determine this behavior before all the computations are made.

3.6 Enhanced Labeling Procedure

The XTI method for the double-pivot network simplex method essentially inherits the operations described in Section 2.4. However, it is called upon twice and sequentially when two arcs leave and enter the basis. Additionally, the reduced cost c_{ij}^{π} is recomputed for the second entering arc to account for the change in node potential values from the threading procedure of the previous XTI method operation. Further research is needed on modifying the XTI method to account for double pivots simultaneously instead of calling the XTI method twice and sequentially. This can potentially reduce the overall computational time of the double-pivoting method.

Other improvements have been made to speed up the time of the double-pivot network simplex method. One of them is in finding the three minimum arcs in both cycles: the minimum overlapping arc, the minimum non-overlapping arc in cycle one, and the minimum non-overlapping arc in cycle two. The next section discusses this.

3.6.1 Overlap and Flow Direction Arrays

Let O_i denote an integer array of length n where i is the node index for each value in O, and n is the total number of nodes in N. Initialize O as $O = \underbrace{[0, 0, 0, \ldots, 0, 0, 0]}_{\text{length } n}$. Then, when constructing cycles one and two, denote O_i for every node i that appears in either cycle as:

$$O_i = \begin{cases} O_i + 1 & \text{if node } i \text{ is in cycle one.} \\ O_i + 2 & \text{if node } i \text{ is in cycle two.} \end{cases}$$

After both cycle constructions, any index i in O should be categorized under one of the following cases:

$$O_i = \begin{cases} 0 & \text{if node } i \text{ is not in cycles one or two.} \\ 1 & \text{if node } i \text{ is in cycle one.} \\ 2 & \text{if node } i \text{ is in cycle two.} \\ 3 & \text{if node } i \text{ is in both cycle one and two.} \end{cases}$$

Instead of investigating cycles one and two independently for the minimum non-overlapping arcs and then investigating both cycles for the minimum overlapping arc as three distinct steps, one can investigate the nodes that appear in O where $O_i > 0$ for a node i. This can be more effective if one collects the visited nodes during both cycle constructions and only visits the indices in O that appear in the collected node list. For each iteration, one can set $O_i = 0$ for every node i visited in previous cycle constructions to reuse them during the next iteration.

Let C_i^1 and C_i^2 denote boolean arrays of length n where i is the node index for each list, and n is the total number of nodes in N. Initialize both as

$$C^1 = \underbrace{[False, False, \ldots, False, False]}_{\text{length } n} \qquad C^2 = \underbrace{[False, False, \ldots, False, False]}_{\text{length } n}$$

where C_i^1 is the direction of flow associated with the nodes in cycle one and their predecessors $p(i)$ at any index i. Likewise, C_i^2 is the direction of flow associated with the nodes in cycle two and their predecessors $p(i)$. When building cycles one and two, respectively, the following is done for any node that appears in C:

$$C = \begin{cases} True & \text{if the arc at node } i \text{ is increasing in flow} \\ False & \text{if the arc at node } i \text{ is decreasing in flow.} \end{cases}$$

Observe that these values do not need to be cleared at each iteration since both C^1 and C^2 should only be called upon at an index i where $O_i > 0$.

3.7 Summary of the Double-Pivot Network Simplex Method

Algorithm 10 presents all the steps of the double-pivot network simplex method considering all implementation details discussed in previous sections. The following section presents an iterative example of the double-pivot network simplex method to solve minimum-cost network flow problems.

Algorithm 10 The Double-Pivot Network Simplex Method.

procedure Double-Pivot Network Simplex Method(V,E)
  Let $M \leftarrow \sum_{(i,j)\in A} |c_{ij}|$
  if $M = 0$ **then**
    $M \leftarrow 1$
  end if
  Let $n \leftarrow |N| + 1$
  Let $p \leftarrow [\underbrace{n, n, n, \ldots, n, n, n}_{\text{length } n-1}, 0]$
  Let $s \leftarrow [\underbrace{1, 1, 1, \ldots, 1, 1, 1}_{\text{length } n-1}, n]$
  Let $e \leftarrow [1, 2, 3, 4, \ldots, n-3, n-2, n-1]$
  Let $t \leftarrow [2, 3, 4, \ldots, n-1, n, n+1, 1]$
  Let $rt \leftarrow [1, 2, 3, 4, \ldots, n-3, n-2, n-1]$
  Let $w \leftarrow [\underbrace{0, 0, 0, \ldots, 0, 0, 0}_{\text{length } n}]$
  Let $O \leftarrow [\underbrace{0, 0, 0, \ldots, 0, 0, 0}_{\text{length } n}]$
  Let $C^1 \leftarrow [\underbrace{False, False, \ldots, False, False}_{\text{length } n}]$
  Let $C^2 \leftarrow [\underbrace{False, False, \ldots, False, False}_{\text{length } n}]$
  Let $d \leftarrow [\underbrace{False, False, False, \ldots, False, False, False}_{\text{length } n-1}, 0]$
  for all $i \in N$ **do**
    $w_i \leftarrow M$ if $b_i > 0$
    $d_i \leftarrow True$ if $b_i > 0$
  end for

```
    Calculate the reduced costs c_ij^π for block pivoting
    while there exists an improving c_ij^π(s) via block pivoting do
        Let entering arc one equal the most improving c_ij^π
        Let entering arc two equal the second most improving c_ij^π
        Call Algorithm 1 for cycle one
        if there exists a second entering arc then
            Call Algorithm 1 for cycle two
            if there is no overlap then
                Double-single pivot both cycles
            else
                Call Algorithm 8 or 9 to calculate the flow change for both cycles
            end if
        else
            Single-pivot cycle one
        end if
        Update the flow(s) for the cycle(s)
        Call Algorithm 2, 3, 4, 5, 6
        Recalculate the reduced costs c_ij^π for block pivoting
    end while
    Return z* = Σ_{(i,j)∈A} c_ij x_ij and the optimal spanning tree basis
end procedure
```

3.8 Iterative Example of the Double-Pivot Network Simplex Method

To illustrate the double-pivot network simplex method, consider **Example** 3.

Example 3 *Consider the minimum-cost network flow problem presented in* ***Figure*** *3.13 and the initial spanning tree rooted at node 6.*

Let $M = 49$ in this case. Improving arcs are selected using the traditional best-arc approach, except for the first iteration, for a better demonstration of the algorithm. Let $\Delta f = [x_1, x_2]$ where x_1 is the change of flow for cycle one and x_2 is the change of flow for cycle two.

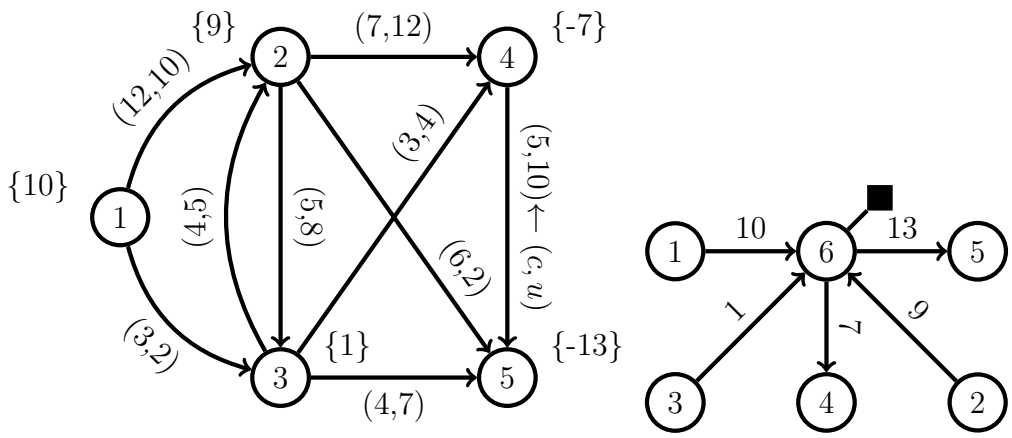
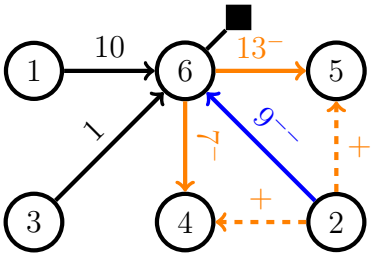


Figure 3.13: Network and initial spanning tree of Example 3.

First, compute the node potential w for all nodes in the spanning tree basis and the reduced cost c_{ij}^{π} for all arcs as shown in **Figure** 3.14. Let $(2, 5)$ and $(2, 4)$ be the entering arcs for this iteration. Observe that entering both arcs into the spanning tree shown in 3.14 induces two cycles: 2-5-6-2 and 2-4-6-2. Furthermore, both cycles are in the same direction and they overlap. Since this is a positive flow case, the two-variable linear program is solved using **Algorithm** 8. The change of flow returned is $\Delta f = [2, 7]$. The flow on both cycles is updated such that the new spanning tree has $x_{2,5} = 2$, $x_{6,5} = 11$, $x_{2,4} = 7$, $x_{6,4} = 0$, and $x_{2,6} = 0$. Therefore, the leaving arcs are $(2, 6)$ and $(2, 5)$ at its upper bound.

Iteration One, $z = 980$



Call **Algorithm** 8, $\Delta f = [2, 7]$

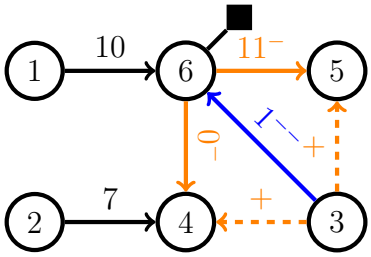
n	**t**	**rt**	**p**	**s**	**e**	**d**	**w**	**o**	C^1	C^2
1	2	6	6	1	1	T	49	0	F	F
2	3	1	6	1	2	T	49	3	F	F
3	4	2	6	1	3	T	49	0	F	F
4	5	3	6	1	4	F	0	2	F	F
5	6	4	6	1	5	F	0	1	F	F
6	1	5	0	6	5	0	0	0	F	F

(i,j)	$w_i - w_j - c_{ij}$	c^{π}_{ij}	***Status***
(1, 2)	49 - 49 - 12	-12	Lower
(1, 3)	49 - 49 - 3	-3	Lower
(2, 3)	49 - 49 - 5	-5	Lower
P^*(2, 4)	49 - 0 - 7	42	Lower
P^*(2, 5)	49 - 0 - 6	43	Lower
(3, 2)	49 - 49 - 4	-4	Lower
(3, 4)	49 - 0 - 3	46	Lower
(3, 5)	49 - 0 - 4	45	Lower
(4, 5)	0 - 0 - 5	-5	Lower

Figure 3.14: Iteration one of **Example** 3.

On the second iteration (**Figure** 3.15), the dual variables and reduced costs are computed. In this case, arcs $(3, 4)$ and $(3, 5)$ enter the basis since they have the largest reduced costs. Notice that two cycles are created: 3-4-6-3 and 3-5-6-3. Both cycles are in the same direction and the change of flow $\Delta f = [0, 1]$ is determined by **Algorithm** 8. The leaving arcs for this iteration are $(6, 4)$ and $(3, 5)$. The flow is updated so that $x_{3,4} = 0$, $x_{6,4} = 0$, $x_{3,5} = 1$, $x_{6,5} = 10$, and $x_{3,6} = 0$ in the next spanning tree.

Iteration Two



Call **Algorithm** 8, $\Delta f = [0, 1]$

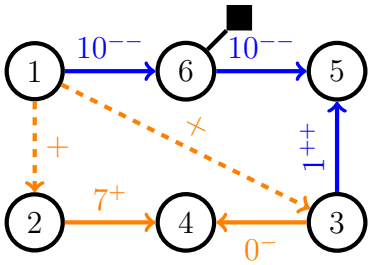
n	**t**	**rt**	**p**	**s**	**e**	**d**	**w**	**o**	C^1	C^2
1	3	6	6	1	1	T	49	0	F	F
2	5	4	4	1	2	T	7	0	F	F
3	4	1	6	1	3	T	49	3	F	F
4	2	3	6	2	2	F	0	1	F	F
5	6	2	6	1	5	F	0	2	F	F
6	1	5	0	6	5	0	0	0	F	F

(i,j)	$w_i - w_j - c_{ij}$	c^{π}_{ij}	***Status***
(1, 2)	49 - 7 - 12	30	Lower
(1, 3)	49 - 49 - 3	-3	Lower
(2, 3)	7 - 49 - 5	-47	Lower
(2, 4)	7 - 0 - 7	0	BV
(2, 5)	7 - 0 - 6	1	Upper
(3, 2)	49 - 7 - 4	38	Lower
P^*(3, 4)	49 - 0 - 3	46	Lower
P^*(3, 5)	49 - 0 - 4	45	Lower
(4, 5)	0 - 0 - 5	-5	Lower

Figure 3.15: Iteration two of **Example** 3.

The process is repeated for the next iterations. The entering arcs in iteration three (**Figure** 3.16) are $(1, 3)$ and $(1, 2)$, thus inducing the following two cycles: 1-3-5-6-1 and 1-2-4-3-5-6-1. Since $\Delta f = [2, 0]$, which is determined by **Algorithm** 8, then arcs $(1, 3)$ at its upper bound and $(3, 4)$ leave the basis.

Iteration Three



Call **Algorithm** 8, $\Delta f = [2, 0]$

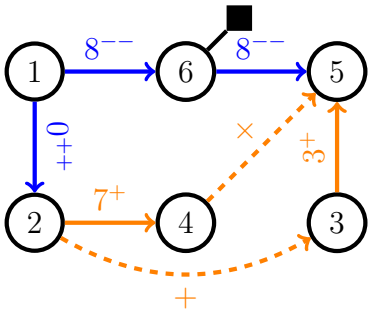
n	**t**	**rt**	**p**	**s**	**e**	**d**	**w**	**o**	C^1	C^2
1	5	6	6	1	1	T	49	3	F	F
2	6	4	4	1	2	T	8	2	F	T
3	4	5	5	3	2	T	4	3	T	T
4	2	3	3	2	2	F	1	2	F	F
5	3	1	6	4	2	F	0	3	F	F
6	1	2	0	6	2	0	0	0	F	F

(i,j)	$w_i - w_j - c_{ij}$	c_{ij}^{π}	***Status***
$P^*(1, 2)$	49 - 8 - 12	29	Lower
$P^*(1, 3)$	49 - 4 - 3	42	Lower
(2, 3)	8 - 4 - 5	-1	Lower
(2, 4)	8 - 1 - 7	0	BV
(2, 5)	8 - 0 - 6	2	Upper
(3, 2)	4 - 8 - 4	-8	Lower
(3, 4)	4 - 1 - 3	0	BV
(3, 5)	4 - 0 - 4	0	BV
(4, 5)	1 - 0 - 5	-4	Lower

Figure 3.16: Iteration three of **Example** 3.

For the fourth iteration (**Figure** 3.17), arcs $(2, 3)$ and $(4, 5)$ enter the basis resulting in the following cycles: 2-3-5-6-1-2 and 4-5-6-1-2-4. **Algorithm** returns $\Delta f = [4, 4]$ and the spanning tree is updated. The leaving arcs are $(3, 5)$ and $(6, 5)$ in this case. After the node potentials and reduced costs are updated as depicted in **Figure** 3.18, no improving arcs exist. Therefore, the spanning tree depicted in **Figure** 3.18 is optimal and its corresponding objective function value is $z^* = 259$.

Iteration Four



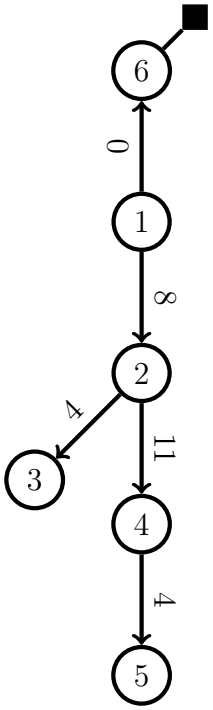
Call **Algorithm** 8, $\Delta f = [4, 4]$

n	**t**	**rt**	**p**	**s**	**e**	**d**	**w**	**o**	C^1	C^2
1	2	6	6	5	5	T	49	3	F	F
2	3	1	1	4	5	F	37	3	T	T
3	4	2	2	1	3	F	32	1	T	F
4	5	3	2	2	5	F	30	2	F	T
5	6	4	4	1	5	F	25	3	F	F
6	1	5	0	6	5	0	0	0	F	F

(i,j)	$w_i - w_j - c_{ij}$	c^{π}_{ij}	***Status***
(1, 2)	49 - 37 - 12	0	BV
(1, 3)	49 - 4 - 3	42	Upper
P^*(2, 3)	37 - 4 - 5	28	Lower
(2, 4)	37 - 30 - 7	0	BV
(2, 5)	37 - 0 - 6	31	Upper
(3, 2)	4 - 37 - 4	-37	Lower
(3, 4)	4 - 30 - 3	-29	Lower
(3, 5)	4 - 0 - 4	0	BV
P^*(4, 5)	30 - 0 - 5	25	Lower

Figure 3.17: Iteration four of **Example** 3.

Final Results, $z = 259$



n	**t**	**rt**	**p**	**s**	**e**	**d**	**w**	**o**	C^1	C^2
1	2	6	6	5	5	T	49	3	F	F
2	3	1	1	4	5	F	37	3	T	T
3	4	2	2	1	3	F	32	1	T	F
4	5	3	2	2	5	F	30	2	F	T
5	6	4	4	1	5	F	25	3	F	F
6	1	5	0	6	5	0	0	0	F	F

(i,j)	$w_i - w_j - c_{ij}$	c_{ij}^{π}	***Status***
(1, 2)	49 - 37 - 12	0	BV
(1, 3)	49 - 32 - 3	14	Upper
(2, 3)	37 - 32 - 5	0	BV
(2, 4)	37 - 30 - 7	0	BV
(2, 5)	37 - 25 - 6	6	Upper
(3, 2)	32 - 37 - 4	-9	Lower
(3, 4)	32 - 30 - 3	-1	Lower
(3, 5)	32 - 25 - 4	3	Upper
(4, 5)	30 - 25 - 5	0	BV

Figure 3.18: Final result of **Example** 3.

4 Computational Results and Analysis

This chapter discusses the computational study performed in this thesis to determine whether the double-pivot network simplex method solves minimum-cost network flow problems faster than the single-pivot network simplex method. The computational study also provides results on the double-pivot network simplex method's performance regarding the number of iterations.

Cython, a programming language that is an extension of Python allowing for highly efficient programming optimization, was used along with Jupyter Notebook to implement the algorithms presented in this thesis[22 4]. Computational experiments tested problems using an HP EliteBook 830 G6 Laptop with an Intel® Core™ i7-8665U 1.90GHz processor and 16GB of RAM.

4.1 NETGEN Benchmark Library

The benchmark data used to test the network simplex methods are from Kovác's LEMON test-suite[23]. The test suite comprises graphs of various sizes as depicted in **Table** 4.1.

Table 4.1: Number of arcs and nodes contained in each NETGEN instance.

NETGEN 8	**NETGEN 9**	**NETGEN 10**
256 Nodes x 2048 Arcs	512 Nodes x 4096 Arcs	1024 Nodes x 8192 Arcs
NETGEN 11	**NETGEN 12**	**NETGEN 13**
2048 Nodes x 16,384 Arcs	4096 Nodes x 32,768 Arcs	8192 Nodes x 65,536 Arcs
NETGEN 14	**NETGEN 15**	**NETGEN 16**
16,384 Nodes x 131,072 Arcs	32,768 Nodes x 262,144 Arcs	65,536 Nodes x 524,288 Arcs
NETGEN 17	**NETGEN 18**	**NETGEN 19**
131,072 Nodes x 1,048,576 Arcs	262,144 Nodes x 2,097,152 Arcs	524,288 Nodes x 4,194,304 Arcs
NETGEN 20	**NETGEN 21**	**NETGEN 22**
1,048,576 Nodes x 8,388,608 Arcs	2,097,152 Nodes x 16,777,216 Arcs	4,194,304 Nodes x 33,554,432 Arcs

Each NETGEN instance in **Table** 4.1 has five problems: A, B, C, D, and E. Therefore, there are 70 benchmark instances in total. Additionally, three sets of testing were performed: alpha testing of the best block sizes for the double-pivoting method; CPU time testing of the single and double-pivoting methods, CPLEX, Gurobi, and NetworkX; and the number of iterations needed to solve every benchmark problem for both the single and double-pivoting methods.

4.2 Alpha Testing

The implementation of both the single and double-pivot network simplex algorithms in this thesis uses the block pivoting rule to reduce the computation time needed to determine entering arcs at each iteration. However, the double-pivoting method

attempts to look for two entering arcs instead of one in each block - but if the algorithm searches in a block and only finds one entering arc, then it accepts only that one improving arc for that iteration.

Both implementations use a block size of $\alpha\sqrt{m}$ where $\alpha \in \mathbb{R}^+$ and m is the total number of arcs in A excluding the artificial arcs. Kiraly and Kovács in 2012 found that a block size where $\alpha = 1$ produces the best results for the single-pivoting method[21]. Since the double-pivoting method takes in twice the arcs per iteration, a natural choice for the double-pivoting method is to set $\alpha = 2$.

To test this hypothesis, the computational study within this thesis solved some of the benchmark instances for various values of α. NETGEN instances 8, 9, and 10 were used since in early development it was found that these small benchmark instances would slow down the double-pivoting method due to false pivoting. Testing was done incrementally (by a value of 0.01), increasing the value of α from 1 until the double-pivoting method outperformed the single-pivoting method in terms of CPU runtime. Testing has shown that the double-pivoting method outperformed the single-pivoting method when $\alpha \in [1.7, 2.3]$, and the best results were obtained when $\alpha = 2$. Consequently, the double-pivoting method was implemented with $\alpha = 2$.

4.3 CPU Runtime Testing

The double-pivoting method outperformed the single-pivoting method for substantially larger problems in terms of CPU runtime. For small instances, the single-pivoting method outperformed the double-pivoting algorithm, and this is suspected due to the number of false pivots the latter method encountered throughout the problems. NETGEN 22 was inconclusive from problems B onwards for both network methods as they hit a 48-hour computational threshold established in this thesis.

Tables 4.2-4.4 contains all the CPU times for the benchmark problems. Because background processes in machines can compromise the CPU times during the benchmarking, each of the five problems within the benchmark instances was run many times, and the results presented in **Tables** 4.2-4.4 are the averages of these runs. For instances NETGEN 08 through 11, each of the five problems was run 100 times. For instances NETGEN 12 through 15, problems were run 50 times. Furthermore, problems within instances NETGEN 16-19 were run 10 times while problems within instances NETGEN 20-22 were run a single time due to the size of the problems. The percentage improvement shown in **Tables** 4.2-4.4 are computed as $1 - \frac{DP}{SP}$ where DP is the CPU time of the double-pivoting method and SP is the CPU time of the single-pivoting method.

Observe that in every single instance, the single and double-pivoting methods outperformed NetworkX but could not compare to the commercial solvers CPLEX and Gurobi. It is worth noting that Gurobi had beaten CPLEX in every instance. Notice that NETGEN 8 through 12 is where the single-pivoting method outperforms the double-pivoting technique by about 8%, on average, and NETGEN 13 through 22 is where the double-pivoting method outperforms the single-pivoting algorithm by approximately 12% on average. **Figures** A.1-A.5 helps better visualize these results, and are shown in Appendix A.

Table 4.2: CPU runtime, in milliseconds, for NETGEN instances 8-12.

NETGEN 08	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	31	64	97	135	162	97.8
Double	29	62	107	155	189	108.4
DoCPLEX	5	11	18	24	30	17.6
Gurobi	3	5	7	10	14	7.8
NetworkX	95	174	267	365	453	270.8
% Improvement Double over Single	6.45%	3.13%	-10.31%	-14.81%	-16.67%	-10.84%

NETGEN 09	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	245	328	425	518	606	424.4
Double	281	373	480	604	696	486.8
DoCPLEX	44	58	71	85	99	71.4
Gurobi	20	27	33	38	42	32
NetworkX	713	930	1231	1501	1768	1228.6
% Improvement Double over Single	-14.69%	-13.72%	-12.94%	-16.60%	-14.85%	-14.70%

NETGEN 10	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	838	1046	1232	1471	1714	1260.2
Double	931	1154	1354	1607	1861	1381.4
DoCPLEX	125	143	162	184	204	163.6
Gurobi	57	67	79	93	106	80.4
NetworkX	2404	2972	3602	4200	4836	3602.8
% Improvement Double over Single	-11.10%	-10.33%	-9.90%	-9.25%	-8.58%	-9.62%

NETGEN 11	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	2237	2843	3373	3856	4414	3344.6
Double	2295	2942	3549	4147	4683	3523.2
DoCPLEX	238	276	314	348	381	311.4
Gurobi	167	235	308	368	436	302.8
NetworkX	6469	8199	9917	11514	13001	9820
% Improvement Double over Single	-2.59%	-3.48%	-5.22%	-7.55%	-6.09%	-5.34%

NETGEN 12	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	1763	1923	3737	5885	7911	4243.8
Double	1816	1908	3849	5869	7842	4256.8
DoCPLEX	97	124	283	371	421	259.2
Gurobi	201	270	539	790	837	527.4
NetworkX	4671	5858	11313	16772	24114	12545.6
% Improvement Double over Single	-3.01%	0.78%	-3.00%	0.27%	0.87%	-0.31%

Table 4.3: CPU runtime, in milliseconds, for NETGEN instances 13-17.

NETGEN 13	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	7404	12474	18046	24066	30654	18528.8
Double	5853	10530	15883	23390	31722	17475.6
DoCPLEX	205	613	802	1210	1382	842.4
Gurobi	377	691	1052	1443	1772	1067
NetworkX	23227	39655	56466	73973	90996	56863.4
% Improvement Double over Single	20.95%	15.58%	11.99%	2.81%	-3.48%	5.68%

NETGEN 14	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	46681	65916	80730	95966	110125	79883.6
Double	48910	65143	79524	94434	108281	79258.4
DoCPLEX	2292	3296	4144	4991	5775	4099.6
Gurobi	2428	3181	3762	4343	4908	3724.4
NetworkX	138143	186692	231398	275879	322285	230879.4
% Improvement Double over Single	-4.77%	1.17%	1.49%	1.60%	1.67%	0.78%

NETGEN 15	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	58062	117176	177080	241780	294620	177743.6
Double	53021	80462	131439	174598	225643	133032.6
DoCPLEX	3264	5571	8050	9933	11964	7756.4
Gurobi	1914	3958	5646	7121	8646	5457
NetworkX	130504	287851	441287	619805	747594	445408.2
% Improvement Double over Single	8.68%	31.33%	25.77%	27.79%	23.41%	25.15%

NETGEN 16	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	172270	311380	455025	575176	715733	445916.8
Double	114312	170064	227589	295756	380560	237656.2
DoCPLEX	7014	11816	16743	21450	26049	16614.4
Gurobi	5665	9510	13283	17284	20986	13345.6
NetworkX	399855	703089	1025871	1393218	1724544	1049315.4
% Improvement Double over Single	33.64%	45.38%	49.98%	48.58%	46.83%	46.70%

NETGEN 17	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	1093722	1460181	1857008	2233305	2626483	1854139.8
Double	752622	1139723	1554850	1931334	2137485	1503202.8
DoCPLEX	39749	52930	66824	80459	95086	67009.6
Gurobi	30684	40823	50256	59562	69107	50086.4
NetworkX	2878612	3994127	5118681	6150404	7237863	5075937.4
% Improvement Double over Single	31.19%	21.95%	16.27%	13.52%	18.62%	18.93%

Table 4.4: CPU runtime, in milliseconds, for NETGEN instances 18-22.

NETGEN 18	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	3753203	4851626	6002848	7113914	8227601	5989838.4
Double	3125211	4272068	5471412	6594556	7728895	5438428.4
DoCPLEX	136305	180306	222907	266498	310226	223248.4
Gurobi	94704	119238	144956	172615	200220	146346.6
NetworkX	9872275	12706944	15309654	18007831	20470115	15273363.8
% Improvement Double over Single	16.73%	11.95%	8.85%	7.30%	6.06%	9.21%

NETGEN 19	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	1161318	3293757	7186112	10093936	13006177	6948260
Double	1118410	3178282	6884711	10062284	12824091	6813555.6
DoCPLEX	475836	158654	327772	478914	635814	415398
Gurobi	264373	64695	131873	192984	251329	181050.8
NetworkX	2996906	9792062	18437868	28633543	37589834	19490042.6
% Improvement Double over Single	3.69%	3.51%	4.19%	0.31%	1.40%	1.94%

NETGEN 20	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	21959593	30741853	39182776	48034459	56816386	39347013.4
Double	21351020	29801298	37641499	46037648	54328431	37831979.2
DoCPLEX	1019536	1408870	1820416	2216601	2618110	1816706.6
Gurobi	398832	541414	679802	828331	977651	685206
NetworkX	54306195	70343004	85929587	102515221	118930377	86404876.8
% Improvement Double over Single	2.77%	3.06%	3.93%	4.16%	4.38%	3.85%

NETGEN 21	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	24313672	23411156	49691492	49284175	75516235	44443346
Double	23949972	22667257	48207037	46159168	73960077	42988702.2
DoCPLEX	1031472	1055617	2123584	2455442	3477647	2028752.4
Gurobi	351023	337298	692694	754548	1103169	647746.4
NetworkX	64736731	130087247	> 48hrs	> 48hrs	> 48hrs	142644795.6
% Improvement Double over Single	1.50%	3.18%	2.99%	6.34%	2.06%	3.27%

NETGEN 22	**A**	**B**	**C**	**D**	**E**	**AVERAGES**
Single	153643645	> 48hrs	> 48hrs	> 48hrs	> 48hrs	168968729
Double	147731174	> 48hrs	> 48hrs	> 48hrs	> 48hrs	167786234.8
DoCPLEX	2649300	5448027	8194377	10934995	13694524	8184244.6
Gurobi	853342	1722620	2560526	3397069	4293840	2565479.4
NetworkX	> 48hrs	> 48hrs	> 48hrs	> 48hrs	> 48hrs	> 48hrs
% Improvement Double over Single	3.85%	0.00%	0.00%	0.00%	0.00%	0.70%

4.4 Iterations and Pivot Type Occurrences

The computational study tested the double-pivoting method and the single-pivoting method for their iteration counts. The double-pivoting method solved NETGEN instances (problems A through E) in fewer iterations than the single-pivoting method in every single problem by approximately 50%. Tables 4.5-4.7 present the results. The percentage improvements are computed similarly as in Tables 4.2-4.4.

Observe that the range of 49%∼53% is consistent across the values regardless of problem size. **Tables** 4.5-4.7 contain all the iterative data for each benchmark problem and show the problem breakdown by iteration type for the double-pivoting network simplex method: single-pivots, double-single pivots, double pivots, and false pivots. Additionally, one can see that the larger the problem, the smaller the number of false double-pivots, as shown in the tables. The author believes this may cause the CPU time slowdown in the double-pivoting network simplex method for small instances.

One could reduce the number of false double-pivots by removing the cases where only an overlapping arc leaves the basis from the two-variable casework. In developmental testing, it made the runtime results even worse. Thus, future work could determine if there are methods that can be deployed for smaller problems concerning the double-pivoting method. Additionally, alpha testing will affect these results due to scaling the block size by various amounts. Thus, more testing will be needed to investigate if there is a correlation between the number of iterations, runtime, and alpha sizes.

To summarize, the double-pivoting method outperformed the single-pivoting method by 5% in CPU runtime and gained an advantage towards the larger NETGEN problems where the double-pivoting method outperforms the single-pivoting algorithm by approximately 12% on average. Therefore, based on the results obtained in this thesis, the recommendation is that the double-pivot network simplex method should be used to solve substantially large minimum-cost network flow problems over the single-pivot network simplex method.

Table 4.5: Iteration numbers for NETGEN instances 8-12.

Problem Name		Single Pivot	Double Pivot						
		Iterations	*Iterations*	*% Fewer Iterations*	*Single*	*Double Single*	*Double*	*False Double*	*% False Double*
NETGEN 08	A	1022	417	59.20%	25	162	230	68	29.57%
256 Nodes 2048 Arcs	B	1030	439	57.38%	33	133	273	100	36.63%
	C	1085	595	45.16%	27	152	416	130	31.25%
	D	1158	581	49.83%	24	173	384	121	31.51%
	E	973	481	50.57%	159	24	298	100	33.56%
% Iterations Less:		52.43%		Average % False Pivots:			32.50%		
NETGEN 09	A	2253	1044	53.66%	46	346	652	174	26.69%
512 Nodes 4096 Arcs	B	2214	1035	53.25%	65	351	619	198	31.99%
	C	2189	1053	51.90%	38	353	662	186	28.10%
	D	2227	1232	44.68%	32	381	819	203	24.79%
	E	2150	968	54.98%	56	340	572	178	31.12%
% Iterations Less		51.69%		Average % False Pivots:			28.54%		
NETGEN 10	A	4739	2210	53.37%	86	833	1191	239	20.07%
1024 Nodes 8192 Arcs	B	4338	2044	52.88%	86	838	1120	258	23.04%
	C	3936	1846	53.10%	89	716	1041	257	24.69%
	D	4969	2327	53.17%	116	802	1409	307	21.79%
	E	4854	2224	54.18%	82	794	1348	286	21.22%
% Iterations Less		53.34%		Average % False Pivots:			22.16%		
NETGEN 11	A	7967	3240	59.33%	111	1569	1560	289	18.53%
2048 Nodes 16,384 Arcs	B	9093	4212	53.68%	125	1696	2391	452	18.90%
	C	8136	3997	50.87%	89	1715	2193	417	19.02%
	D	7726	4068	47.35%	128	1659	2281	402	17.62%
	E	8492	3795	55.31%	101	1689	2005	370	18.45%
% Iterations Less		53.31%		Average % False Pivots:			18.50%		
NETGEN 12	A	16652	8244	50.49%	172	3483	4021	568	14.13%
4096 Nodes 32,768 Arcs	B	17442	8017	54.04%	181	3670	4166	522	12.53%
	C	15534	7712	50.35%	232	3408	4072	578	14.19%
	D	17570	7576	56.88%	186	3456	3934	575	14.62%
	E	17770	8179	53.97%	283	3781	3573	542	15.17%
% Iterations Less		53.15%		Average % False Pivots:			14.13%		

Table 4.6: Iteration numbers for NETGEN instances 13-17

Problem Name		Single Pivot	Double Pivot						
		Iterations	*Iterations*	*% Fewer Iterations*	*Single*	*Double Single*	*Double*	*False Double*	*% False Double*
NETGEN 13	A	31859	14371	54.89%	253	7288	6830	782	11.45%
8192 Nodes 65,536 Arcs	B	31339	14006	55.31%	251	7451	6304	764	12.12%
	C	32282	14607	54.75%	262	7545	6800	752	11.06%
	D	31268	14919	52.29%	213	7385	7321	823	11.24%
	E	35291	16685	52.72%	204	7691	8790	968	11.01%
% Iterations Less		53.99%		Average % False Pivots:			11.38%		
NETGEN 14	A	60230	29333	51.30%	364	15370	13599	1263	9.29%
16,384 Nodes 131,072 Arcs	B	62126	28203	54.60%	362	15224	12617	1202	9.53%
	C	61341	27619	54.97%	307	15147	12165	1230	10.11%
	D	62846	28807	54.16%	427	15229	13151	1260	9.58%
	E	61140	28120	54.01%	379	15384	12357	1261	10.20%
% Iterations Less		53.81%		Average % False Pivots:			9.74%		
NETGEN 15	A	120967	59512	50.80%	483	32793	24543	1693	6.90%
32,768 Nodes 262,144 Arcs	B	119188	56638	52.48%	524	32182	23932	1767	7.38%
	C	120439	60228	49.99%	452	32482	25463	1831	7.19%
	D	124413	57260	53.98%	574	31930	24756	1742	7.04%
	E	122327	58310	52.33%	521	31546	24612	1631	6.63%
% Iterations Less		51.92%		Average % False Pivots:			7.03%		
NETGEN 16	A	224368	108545	51.62%	631	67340	38229	2345	6.13%
65,536 Nodes 524,288 Arcs	B	234563	108720	53.65%	673	68696	39351	2495	6.34%
	C	239246	105161	56.04%	641	63840	38334	2346	6.12%
	D	212214	98884	53.40%	799	65260	32825	2245	6.84%
	E	233185	109351	53.11%	808	66215	42328	2583	6.10%
% Iterations Less		53.57%		Average % False Pivots:			6.31%		
NETGEN 17	A	430380	204520	52.48%	1176	134773	68571	3592	5.24%
131,072 Nodes 1,048,576 Arcs	B	425039	211360	50.27%	1253	140177	69930	3885	5.56%
	C	451727	218227	51.69%	1078	141718	75431	4012	5.32%
	D	425820	206933	51.40%	1123	134317	71493	3862	5.40%
	E	441602	212553	51.87%	1178	137882	73493	3673	5.00%
% Iterations Less		51.54%		Average % False Pivots:			5.30%		

Table 4.7: Iteration numbers for NETGEN instances 18-22

Problem Name		**Single Pivot**	**Double Pivot**						
		Iterations	*Iterations*	*% Fewer Iterations*	*Single*	*Double Single*	*Double*	*False Double*	*% False Double*
NETGEN 18	A	858857	427353	50.24%	1335	292074	133944	5529	4.13%
	B	835788	417235	50.08%	1749	284713	130773	5398	4.13%
262,144 Nodes	C	862589	429266	50.24%	1607	292381	135278	5442	4.02%
2,097,152 Arcs	D	839578	411146	51.03%	1903	286162	123081	5259	4.27%
	E	857899	417443	51.34%	1635	294578	121230	5583	4.61%
% Iterations Less		50.59%		Average % False Pivots:			4.23%		
NETGEN 19	A	1729660	842714	51.28%	2308	578783	261623	7832	2.99%
	B	1659379	826596	50.19%	2482	576633	247481	8087	3.27%
524,288 Nodes	C	1822304	904988	50.34%	2509	610462	292017	8567	2.93%
4,194,304 Arcs	D	1652555	814204	50.73%	2278	596512	215414	7768	3.61%
	E	1674124	830166	50.41%	2348	598287	229531	7920	3.45%
% Iterations Less		50.59%		Average % False Pivots:			3.25%		
NETGEN 20	A	3404357	1709210	49.79%	3020	1216637	489553	11904	2.43%
	B	3373116	1678670	50.23%	2819	1216408	459443	11385	2.48%
1,048,576 Nodes	C	3244738	1645593	49.28%	3578	1225397	416618	10612	2.55%
8,388,608 Arcs	D	3364363	1689285	49.79%	3433	1219114	466738	12382	2.65%
	E	3304945	1648457	50.12%	3687	1228480	416290	10486	2.52%
% Iterations Less		49.84%		Average % False Pivots:			2.53%		
NETGEN 21	A	6583414	3287272	50.07%	4666	2491555	791051	16104	2.04%
	B	6688396	3354324	49.85%	6239	2515462	832623	16520	1.98%
2,097,152 Nodes	C	6590795	3208409	51.32%	4861	2503235	700313	15249	2.18%
16,777,216 Arcs	D	6574608	3262824	50.37%	3770	2471077	787977	16159	2.05%
	E	6711856	3378319	49.67%	4827	2564609	808883	15919	1.97%
% Iterations Less		50.25%		Average % False Pivots:			2.04%		
NETGEN 22	A	13129550	6653175	49.33%	6015	5104422	1542738	23372	1.51%
	B	13127413	6525605	50.29%	8743	5174089	1342773	21501	1.60%
4,194,304 Nodes	C	13499812	6758496	49.94%	6963	5242587	1508946	22527	1.49%
33,554,432 Arcs	D	13453430	6654379	50.54%	7321	5173238	1473820	21930	1.49%
	E	13320597	7010351	47.37%	7957	5483920	1518474	21839	1.44%
% Iterations Less		49.49%		Average % False Pivots:			1.51%		

5 Conclusion and Future Research

This thesis created the double-pivot network simplex method, the first multidimensional search algorithm to solve minimum-cost network flow problems if presented as a network. The traditional network simplex method, also referred to as the single-pivot network simplex method in this thesis, moves from one spanning tree basis to another spanning tree basis by exchanging exactly one arc per iteration. In comparison, the double-pivot network simplex method can exchange up to two arcs into the spanning tree basis instead of the one-arc approach. This can be done by solving a two-variable linear program that determines the leaving arcs. Due to the special structure of these two-variable problems, this thesis also presents a procedure to quickly solve them.

The network methods in this thesis were implemented using Barr et al.'s data labeling techniques to attempt to bring them closer in terms of average runtime with commercial network optimization solvers such as CPLEX and Gurobi, and the current Python open-source solver NetworkX. Comparisons of the analyzed algorithms were measured by average CPU time in milliseconds and number of pivots using a set of benchmark problems from the NETGEN test suite stored by Kovács.

Computational experiments showed that the double-pivoting method solved some benchmark minimum-cost network flow problems using approximately 50% fewer iterations, on average, than the single-pivoting method. Regarding CPU time, the double-pivoting method outperformed the single-pivot algorithm by about 12% in instances NETGEN 13 through 22. The single-pivoting method solved instances NETGEN 08 through 12 faster than the double-pivoting method by approximately 8%. When averaging all instances, the double-pivoting method proposed in this thesis is over 5% faster than the single-pivoting method. Additionally, the single and double-pivoting methods outperformed NetworkX but could not compare to the commercial solvers CPLEX and Gurobi.

Future work for the double-pivoting method includes exploring alpha testing to fine-tune the ranges of α for block pivoting, as adjusting values with smaller increments may provide further insight into the fragility of the block pivoting approach for the double-pivoting method. Additionally, exploring other pivoting procedures such as the best pivoting arc rule, first pivoting arc rule, candidate list, and alternating candidate list is a future research topic.

Another research avenue is extending the XTI method to split the spanning tree basis into three sub-trees to reduce the computation time of traversing and updating node potentials and reverse sub-trees on that basis. This is not as trivial as it sounds, as there exists difficulties in understanding how the predecessor path and overlap affect the sub-trees before disconnecting, and how they also affect the reattachment process.

Exploring parallelization for various components of the double-pivot network simplex method is also a future research topic. Notice that the XTI method cheapens discovering cycles with entering arcs. Therefore, in a parallel system, one can enter all possible eligible arcs in an iteration and only multi-single pivot those cycles that do not overlap. For cycles that do overlap, one can deploy the two-variable methods to solve if it is only a two-cycle overlap. Furthermore, the double-pivot network simplex method solved minimum-cost network flow problems faster than the single-pivot network simplex method, what about a triple-pivoting approach? Quadruple-pivoting? Quintuple-pivoting? n-pivoting?

References

[1] Armstrong, R. D. and Jin, Z. (1997). A new strongly polynomial dual network simplex algorithm. *Mathematical Programming*, 78:131–148.

[2] Barr, R., Glover, F., and Klingman, D. (1979). Enhancements of spanning tree labeling procedures for network optimization. *INFOR: Information Systems and Operational Research*, 17(1):16–34.

[3] Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D. (2011). *Linear Programming and Network Flows*. John Wiley & Sons.

[4] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39.

[5] Boggs, P. T., Domich, P. D., Donaldson, J. R., and Witzgall, C. (1989). Algorithmic enhancements to the method of centers for linear programming problems. *ORSA Journal on Computing*, 1(3):159–171.

[6] Chandrasiri, A. and Samarathunge, D. (2017). Application of minimum cost flow problem: A case study of crown distributors in Kegalle, Sri Lanka. *International Journal of Scientific & Engineering Research*, 8(1):1850–1853.

[7] Cui, J., An, S., Zhao, M., et al. (2014). A generalized minimum cost flow model for multiple emergency flow routing. *Mathematical Problems in Engineering*, 2014.

[8] Dantzig, G. B. (1948). Programming in a linear structure. *Bulletin of the American Mathematical Society*, 54(11): 1074–1074.

[9] Dantzig, G. B. (1951). Application of the simplex method to a transportation problem. *Activity Analysis and Production and Allocation*, Wiley.

[10] Dantzig, G. B. (1956). *The Simplex Method.* RAND Corporation, Santa Monica, CA.

[11] Dewil, R., Vansteenwegen, P., Cattrysse, D., and Van Oudheusden, D. (2015). A minimum cost network flow model for the maximum covering and patrol routing problem. *European Journal of Operational Research*, 247(1):27–36.

[12] Domich, P. D., Boggs, P. T., Rogers, J. E., and Witzgall, C. (1991). Optimizing over three-dimensional subspaces in an interior-point method for linear programming. *Linear Algebra and its Applications*, 152:315–342.

[13] Dyer, M. E. (1984). Linear time algorithms for two-and three-variable linear programs. *SIAM Journal on Computing*, 13(1):31–45.

[14] Glover, F., Klingman, D., and Stutz, J. (1974). Augmented threaded index method for network optimization. *INFOR: Information Systems and Operational Research*, 12(3):293–298.

[15] Gong, T., Ju, F., and Bu, D. (2024). Accurate prediction of RNA secondary structure including pseudoknots through solving minimum-cost flow with learned potentials. *Communications Biology*, 7(1):297.

[16] Grigoriadis, M. D. (1986). An efficient implementation of the network simplex method. *Springer*, Netflow at Pisa: 83–111.

[17] Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference*, pages 11 – 15.

[18] Jamrunroj, P. and Boonperm, A.-a. (2021). *A New Technique for Solving a 2-Dimensional Linear Program by Considering the Coefficient of Constraints*, pages 276–286. Springer.

[19] Karmarkar, N. and Ramakrishnan, K. (1985). Further developments in the new polynomial-time algorithm for linear programming. *ORSA/TIMES National Meeting, Boston*.

[20] Karp, R. M. (2008). George dantzig's impact on the theory of computation. *Discrete Optimization*, 5(2):174–185. In Memory of George B. Dantzig.

[21] Király, Z. and Kovács, P. (2012). Efficient implementations of minimum-cost flow algorithms. *arXiv preprint arXiv:1207.6381*.

[22] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D.,

Abdalla, S., and Willing, C. (2016). *Jupyter Notebooks – a publishing format for reproducible computational workflows*. IOS Press.

[23] Kovács, P. (2015). Benchmark data for the minimum-cost flow problem. *LEMON. Library for Efficient Modeling and Optimization in Networks*.

[24] Lotero, S., Androulakis, V., Khaniani, H., Hassanalian, M., Shao, S., and Roghanchi, P. (2024). Optimizing fire emergency evacuation routes in underground coal mines: A lightweight network flow approach. *Tunnelling and Underground Space Technology*, 146:105637.

[25] Megiddo, N. (1983). Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776.

[26] Orlin, J. B. (1997). A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78:109–129.

[27] Orlin, J. B., Plotkin, S. A., and Tardos, É. (1993). Polynomial dual network simplex algorithms. *Mathematical Programming*, 60(1):255–276.

[28] Santos, L.-R., Villas-Bôas, F., Oliveira, A. R., and Perin, C. (2019). Optimized choice of parameters in interior-point methods for linear programming. *Computational Optimization and Applications*, 73:535–574.

[29] Shamos, M. I. and Hoey, D. (1976). Geometric intersection problems. *IEEE*, 17th Annual Symposium on Foundations of Computer Science (sfcs 1976): 208–215.

[30] Tarjan, R. E. (1997). Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. *Mathematical Programming*, 78(2):169–177.

[31] Vitor, F. (2018). The ratio algorithm to solve the optimal basis of two constraint linear programs. *Institute of Industrial and Systems Engineers (IISE)*, IIE Annual Conference. Proceedings: 1949–1954.

[32] Vitor, F. (2023). One vs. two vs. multidimensional searches for optimization methods. *IGI Global*, Encyclopedia of Data Science and Machine Learning: 2400–2419.

[33] Vitor, F. and Easton, T. (2018a). The double pivot simplex method. *Mathematical Methods of Operations Research*, 87:109–137.

[34] Vitor, F. and Easton, T. (2018b). A two-dimensional search primal affine scaling interior point algorithm for linear programs. *Institute of Industrial and Systems Engineers (IISE)*, IIE Annual Conference. Proceedings: 1961–1966.

[35] Vitor, F. and Easton, T. (2022). Projected orthogonal vectors in two-dimensional search interior point algorithms for linear programming. *Computational Optimization and Applications*, 83(1):211–246.

[36] Vitor, F. T. (2019). *Two-dimensional search algorithms for linear programming.* Ph.D. Dissertation. Kansas State University.

[37] Yang, Y. and Vitor, F. (2021). (in press). a double-pivot degenerate-robust simplex algorithm for linear programming. *International Journal of Operational Research.*

[38] Zadeh, N. (1973). A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5:255–266.

[39] Zou, L., Yu, C., and Dresner, M. (2013). The application of inventory transshipment modeling to air cargo revenue management. *Transportation Research Part E: Logistics and Transportation Review*, 57:27–44. Selected Papers from the 15th ATRS Conference, Sydney 2011.

Appendices

Appendix A CPU Runtime Comparison

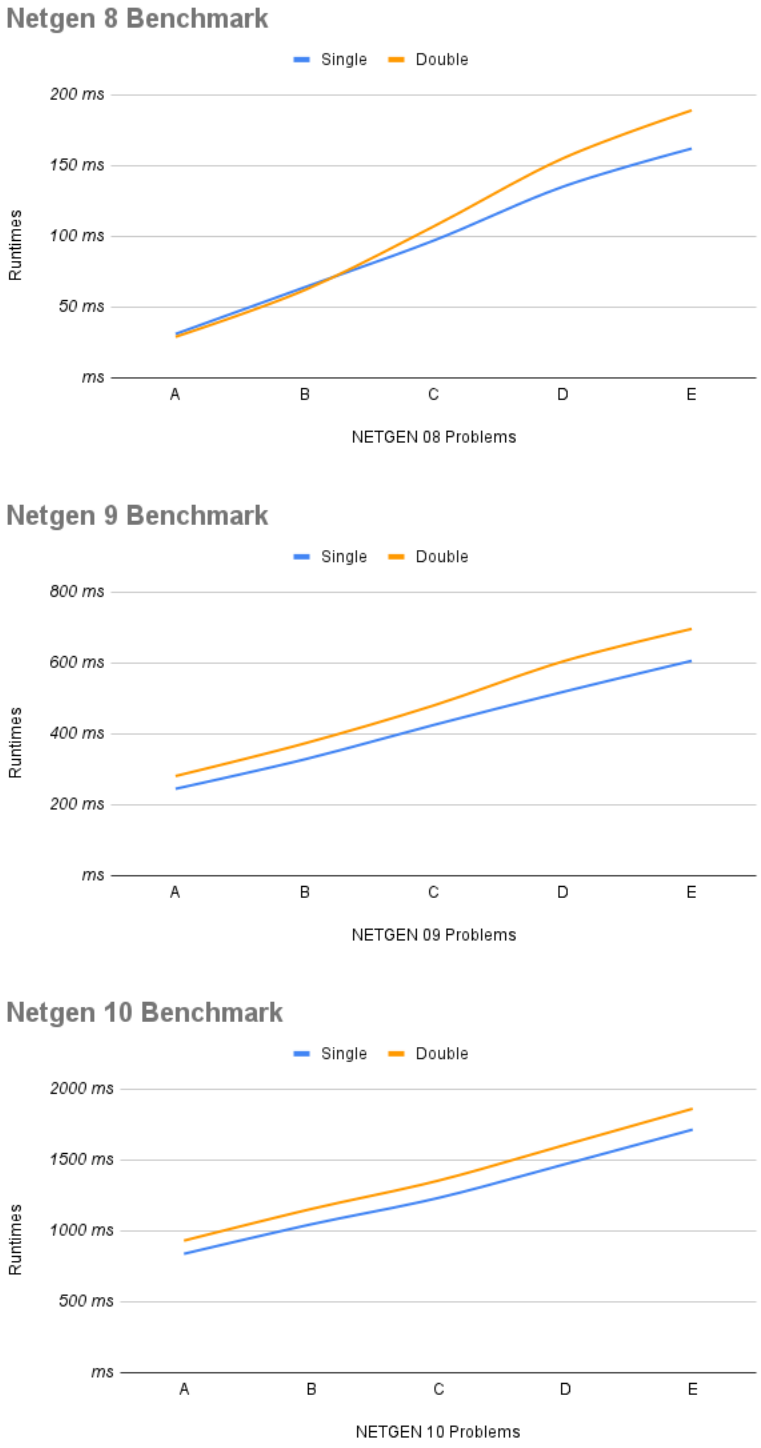


Figure A.1: Performance graphs of the single-pivot network simplex algorithm versus the double-pivot network simplex algorithm for NETGEN instances 8-10.

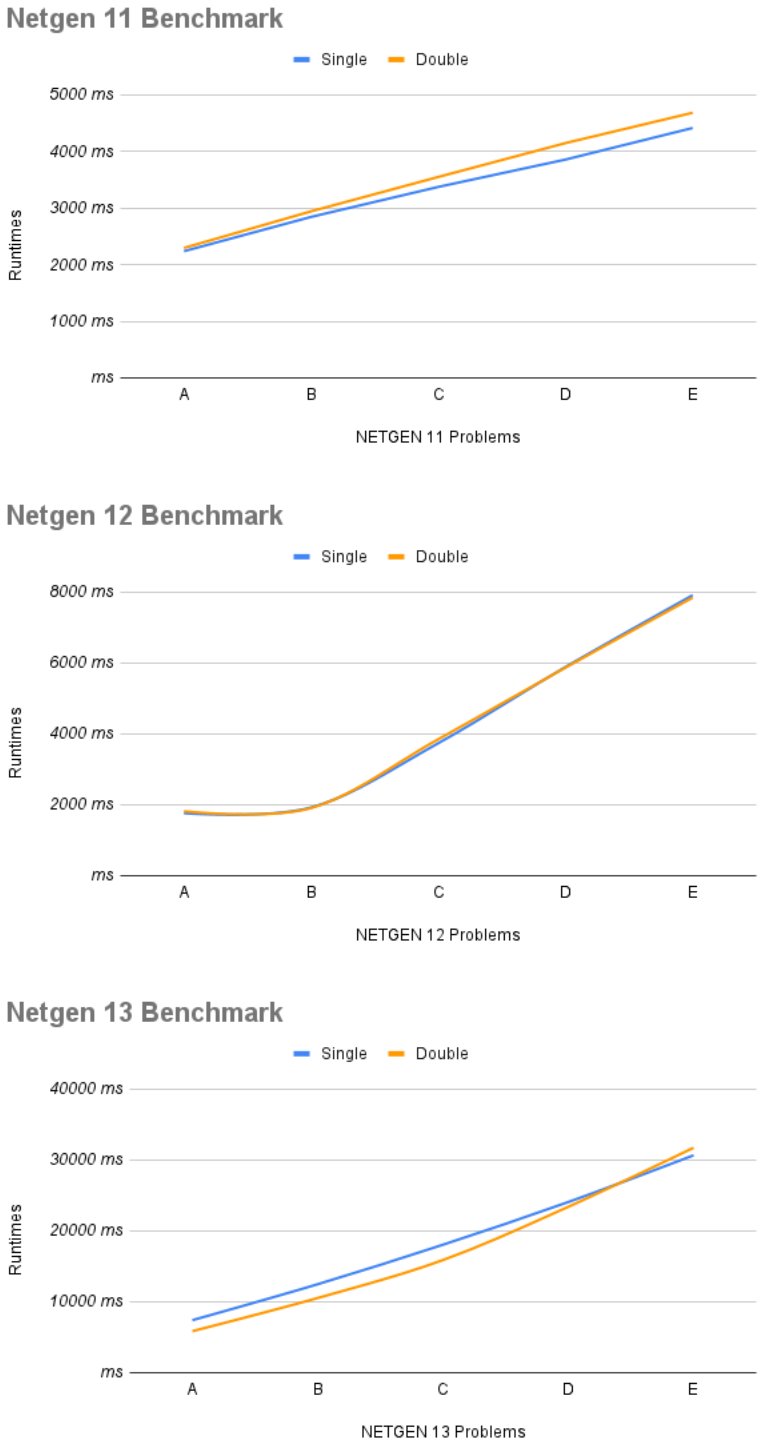


Figure A.2: Performance graphs of the single-pivot network simplex algorithm versus the double-pivot network simplex algorithm for NETGEN instances 11-13.

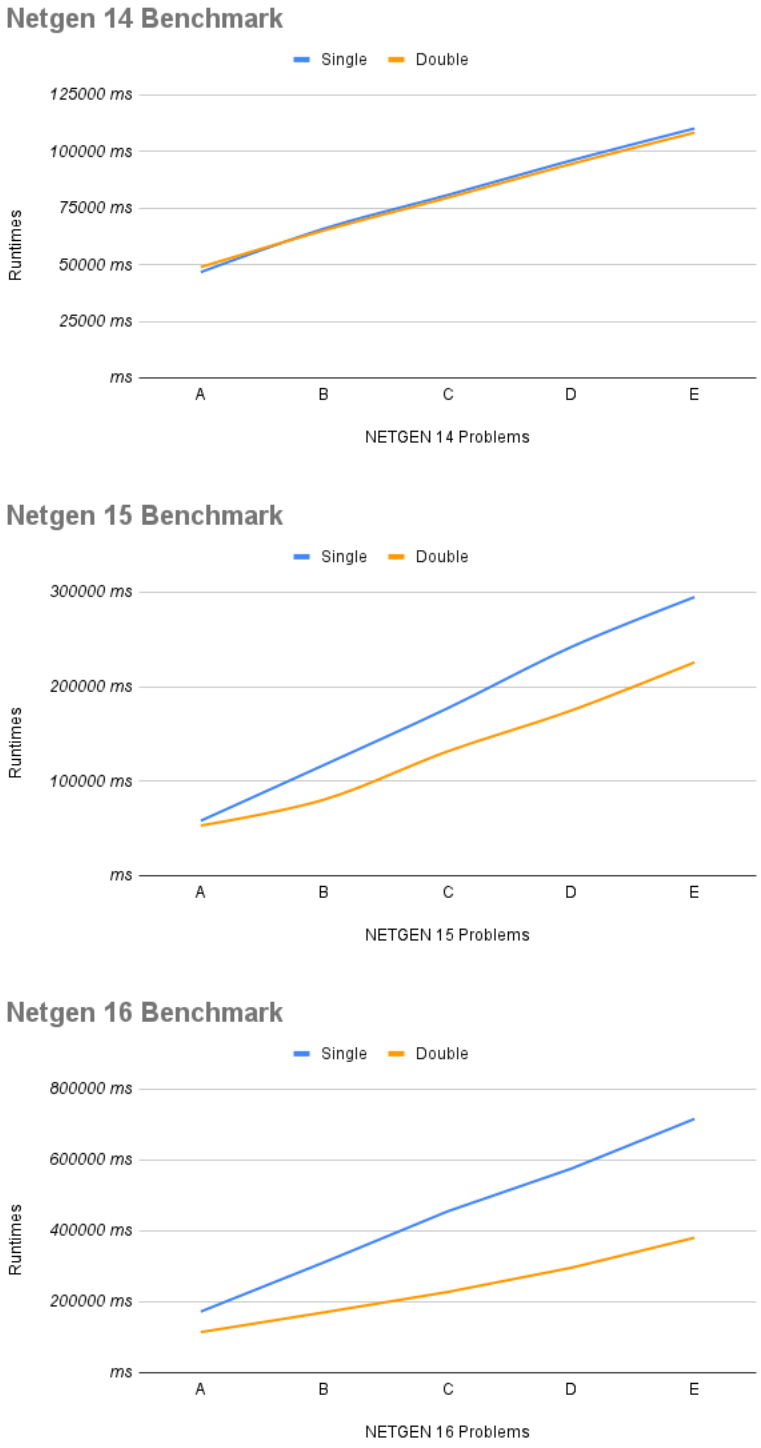


Figure A.3: Performance graphs of the single-pivot network simplex algorithm versus the double-pivot network simplex algorithm for NETGEN instances 14-16.

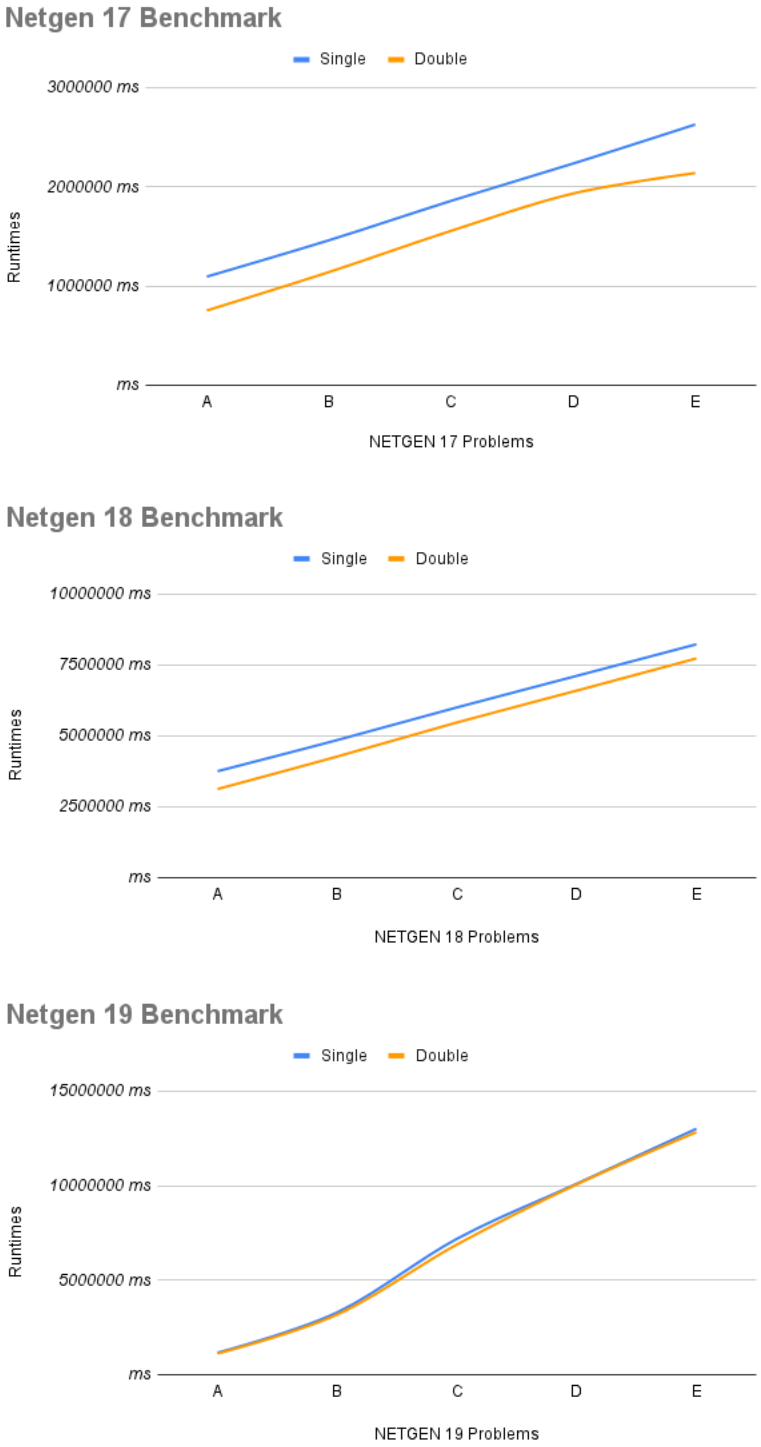


Figure A.4: Performance graphs of the single-pivot network simplex algorithm versus the double-pivot network simplex algorithm for NETGEN instances 17-19.

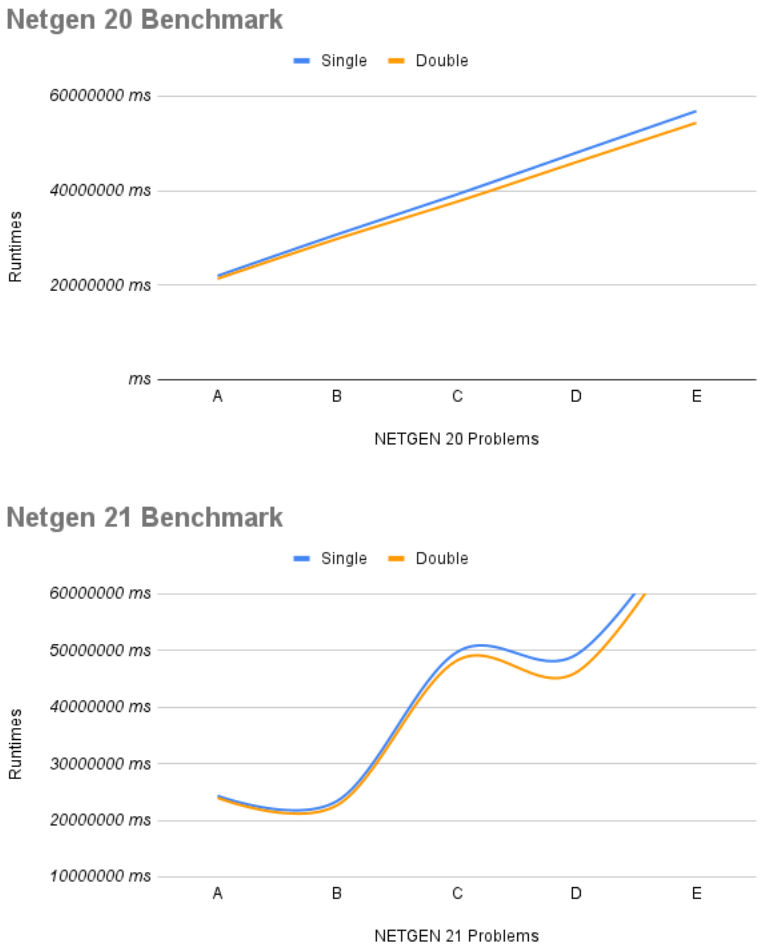


Figure A.5: Performance graphs of the single-pivot network simplex algorithm versus the double-pivot network simplex algorithm for NETGEN instances 20-21.