A New Heuristic Algorithm for Accuracy and Computational Efficiency for Solving Minimum Cost Flow Problems

Timothy Michael Chávez
Old Dominion University, chaveztm@gmail.com

Follow this and additional works at: https://digitalcommons.odu.edu/msve_etds

Part of the Computational Engineering Commons

Recommended Citation
https://digitalcommons.odu.edu/msve_etds/62

This Thesis is brought to you for free and open access by the Computational Modeling & Simulation Engineering at ODU Digital Commons. It has been accepted for inclusion in Computational Modeling & Simulation Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.
A NEW HEURISTIC ALGORITHM FOR ACCURACY AND COMPUTATIONAL EFFICIENCY FOR SOLVING MINIMUM COST FLOW PROBLEMS

by

Timothy Michael Chávez
B.G.S. May 1992, University of Nebraska at Omaha

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

COMPUTATIONAL MODELING AND SIMULATION ENGINEERING

OLD DOMINION UNIVERSITY
August 2021

Approved by:
Duc Thai Nguyen (Director)
Yuzhong Shen (Member)
Hong Yang (Member)
Marsha Sosonkina (Member)
ABSTRACT

A NEW HEURISTIC ALGORITHM FOR ACCURACY AND COMPUTATIONAL EFFICIENCY FOR SOLVING MINIMUM COST FLOW PROBLEMS

Timothy Michael Chávez
Old Dominion University, 2021
Director: Dr. Duc Thai Nguyen

While the minimum cost flow (MCF) problems have been well documented in many publications, due to its broad applications, little or no effort has been devoted to explaining the algorithms for identifying loop formation and computing the θ value needed to solve MCF network problems. This paper proposes efficient numerical procedures and MATLAB computer implementation for computing the θ value. Furthermore, this paper also proposes a mixed heuristic, shortest path (SP) Chavez-Nguyen (or Chayen) Algorithm in Phase 1 (to obtain the basic feasible solution) and, either the conventional MCF or MATLAB’s built-in Linprog() function, in Phase 2 (to obtain the optimal final solution), for a given network problem. Several academic and real-life network problems have been solved to validate the proposed algorithms; the numerical results obtained by the new heuristic code has been compared with the built-in MATLAB Linprog() function (Simplex algorithm) and with the conventional method (where the classical MCF algorithm is applied in both Phases 1 and 2); the results of which are used to validate both the accuracy and computational (time) efficiency of the proposed mixed/hybrid algorithm.
DEDICATION

I am dedicating this thesis to three beloved individuals who have meant and continue to mean so much to me. First, to my mother, Virginia L. Chávez whose love for me knows no bounds and who taught me the value of hard work and the written word.

Next, although no longer of this world, his memory continues to regulate my life; my father, Alex J. Chávez, whose kindness and patience continues to influence me on a daily basis.

And last, but not least, I am dedicating this thesis to my loving wife, Lara M. Arnold. She continues to provide moral and emotional support as well as a loving guidance I can get from no one else.

ACKNOWLEDGMENTS

There are many people who have contributed to the successful completion of this thesis. I extend many, many thanks to my committee members for their patience and hours of guidance on my research and editing of this manuscript. I would also like to express my appreciation to Indian Institute of Technology’s (Madras campus) Professor G. Srinivasan for his helpful comments [ref 1] during the early phase of this work. Also, a heartfelt thanks goes to Mr. David M. Gagliano who also provided constructive feedback leading to the improved, final product. Further, the untiring efforts of my thesis advisor, Dr. Duc Thai Nguyen, deserves special recognition; his continued encouragement pushed me through several roadblocks that allowed me to complete this work.
# NOMENCLATURE

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>arc</td>
<td>The link or edge between two nodes</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>basic link</td>
<td>A link currently contained in the solution; active links</td>
</tr>
<tr>
<td>BFS</td>
<td>Basic Feasible Solution</td>
</tr>
<tr>
<td>$b_x$</td>
<td>The node $x$ supply/demand value</td>
</tr>
<tr>
<td>Chayen algorithm</td>
<td>Chavez-Nguyen algorithm to solve Minimum Cost Flow problems</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>$C_{xy}$</td>
<td>The cost along the link from node $x$ to node $y$</td>
</tr>
<tr>
<td>dead link</td>
<td>A link connected to a dead node</td>
</tr>
<tr>
<td>dead node</td>
<td>A node that with only one or no link</td>
</tr>
<tr>
<td>Eqs.</td>
<td>Equations</td>
</tr>
<tr>
<td>equilibrium</td>
<td>The state in which Link Flow In + Supply = Link Flow Out - Demand</td>
</tr>
<tr>
<td>Iteration Matrix</td>
<td>Matrix that contains the current Chayen solution for a particular iteration</td>
</tr>
<tr>
<td>LHS</td>
<td>Left Hand Side</td>
</tr>
<tr>
<td>link flow</td>
<td>The amount of product to flow across a link</td>
</tr>
<tr>
<td>link status</td>
<td>Defines sign and direction of link flow along the loop</td>
</tr>
<tr>
<td>Linprog()</td>
<td>A built-in MATLAB linear program solver</td>
</tr>
<tr>
<td>LIP</td>
<td>Linear Integer Programming</td>
</tr>
<tr>
<td>loop-formation</td>
<td>The closed set of links needed to calculate the value of theta</td>
</tr>
<tr>
<td>loose node</td>
<td>The node at the end of the loop-formation</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MCF</td>
<td>Minimum Cost Flow</td>
</tr>
<tr>
<td>MCS</td>
<td>Monte Carlo Simulation</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>network</td>
<td>A set of nodes and links associated with a MCF problem</td>
</tr>
<tr>
<td>non-basic link</td>
<td>A link that is considered a solution candidate; non-active links</td>
</tr>
<tr>
<td>OFS</td>
<td>Optimal Final Solution</td>
</tr>
<tr>
<td>Rank/Path Matrix</td>
<td>Matrix containing prioritized rank/path from supply to demand nodes</td>
</tr>
<tr>
<td>RHS</td>
<td>Right Hand Side</td>
</tr>
<tr>
<td>Simplex</td>
<td>A popular algorithm for solving linear preprogramming problems</td>
</tr>
<tr>
<td>sink nodes</td>
<td>A node that has a demand value for product</td>
</tr>
<tr>
<td>SCL</td>
<td>Supply Chain Logistics</td>
</tr>
<tr>
<td>source nodes</td>
<td>A node that a supply value for product</td>
</tr>
<tr>
<td>SP</td>
<td>Shortest Path</td>
</tr>
<tr>
<td>$\theta$ – 'theta'</td>
<td>The minimum positive value for flow along the loop-formation</td>
</tr>
<tr>
<td>transshipment nodes</td>
<td>A node that has no supply nor demand value</td>
</tr>
<tr>
<td>unimodal</td>
<td>A flow network solution containing a single value for an optimal solution</td>
</tr>
<tr>
<td>$w_x$</td>
<td>The weight value for node #x</td>
</tr>
<tr>
<td>$X_{xy}$</td>
<td>The link flow along the link from node #x to node #y</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

LIST OF TABLES ................................................................................................................................................. x

LIST OF FIGURES .................................................................................................................................................. xii

Chapter

1. INTRODUCTION ................................................................................................................................................ 1

2. THE MINIMUM COST FLOW ALGORITHM ........................................................................................................ 7
   2.1 Phase 1 – Calculate the Basic Feasible Solution ......................................................................................... 7
   2.2 Phase 2 – Calculate the Optimal Final Solution (OFS) ........................................................................... 20

3. THE CHAYEN ALGORITHM .......................................................................................................................... 22

4. MONTE CARLO SIMULATION ...................................................................................................................... 38

5. SOLVED EXAMPLES ....................................................................................................................................... 44
   5.1 Example #1: 9-Node/14-Link .................................................................................................................... 45
   5.2 Example #2: 13-Node/20-Link .................................................................................................................. 47
   5.3 Example #3: 20-Node/74-Link .................................................................................................................. 49
   5.4 Example #4: 24-Node/76-Link - Sioux Falls ............................................................................................. 51
   5.5 Example #5: 33-Node/104-Link ................................................................................................................. 53
   5.6 Example #6: 50-Node/102-Link – Johnson City ......................................................................................... 55
   5.7 Example #7: 225-Node/375-Link .............................................................................................................. 57
   5.8 Example #8: 900-Node/1500-Link ............................................................................................................ 59

6. CONCLUSION and FUTURE CONSIDERATIONS .......................................................................................... 61

REFERENCES ......................................................................................................................................................... 69

ATTACHMENTS
   A – MATLAB SCRIPTS ................................................................................................................................. 71
      A.1. Conventional MCF Method, Step #1 ...................................................................................................... 72
      A.2. Conventional MCF Method, Step #2 ...................................................................................................... 77
LIST OF TABLES

Table | Page
--- | ---
1. Rank/Path Matrix | 23
2. Initial Blank Iteration Matrix | 24
6. Rank/Path Matrix | 29
7. Initial Blank Iteration Matrix | 30
8. Iteration Matrix [1] | 31
14. Example #1 - Timing and Iteration Results | 46
15. Example #2 - Timing and Iteration Results | 48
16. Example #3 - Timing and Iteration Results | 50
17. Example #4 - Timing and Iteration Results | 52
18. Example #5 - Timing and Iteration Results | 54
19. Example #6 - Timing and Iteration Results | 56
20. Example #7 - Timing and Iteration Results | 58
21. Example #8 - Timing and Iteration Results .......................................................... 60

22. Phase 2, Step 1 Processing Times ............................................................................. 62
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 5-Node and 7-Link Network</td>
<td>3</td>
</tr>
<tr>
<td>2. Addition of Artificial Node and Links</td>
<td>8</td>
</tr>
<tr>
<td>3. Link 3-5, As $\theta$, Enters the Basic Set</td>
<td>11</td>
</tr>
<tr>
<td>4. Observed Loop</td>
<td>12</td>
</tr>
<tr>
<td>5. Dead Links Deleted</td>
<td>14</td>
</tr>
<tr>
<td>6. Status of Loose Nodes #i and #j</td>
<td>16</td>
</tr>
<tr>
<td>7. General Case for Status Selection</td>
<td>16</td>
</tr>
<tr>
<td>8. End of Phase 1, Iteration 1 (Cost = 12)</td>
<td>19</td>
</tr>
<tr>
<td>9. End of Phase 1, Iteration 2 (Cost = 10)</td>
<td>19</td>
</tr>
<tr>
<td>10. End of Phase 1, Iteration 3 (Cost = 0)</td>
<td>20</td>
</tr>
<tr>
<td>11. Feasible Solution to Start Phase 2</td>
<td>21</td>
</tr>
<tr>
<td>12. Simple 9-Node and 14-Link Network</td>
<td>23</td>
</tr>
<tr>
<td>13. Basic Feasible Solution from Phase 1</td>
<td>27</td>
</tr>
<tr>
<td>14. Basic Feasible Solution from Phase 1 Using Conventional MCF Method</td>
<td>28</td>
</tr>
<tr>
<td>15. More Complex 13-Node and 20-Link Network</td>
<td>29</td>
</tr>
<tr>
<td>16. Basic Feasible Solution from Phase 1</td>
<td>37</td>
</tr>
<tr>
<td>17. Solution Iterations vs Network Size</td>
<td>39</td>
</tr>
<tr>
<td>18. Processing Time vs Network Size</td>
<td>40</td>
</tr>
<tr>
<td>19. 350 Network Execution Time vs Size (scaled)</td>
<td>41</td>
</tr>
<tr>
<td>20. Wall-Clock Execution Time</td>
<td>43</td>
</tr>
<tr>
<td>21. CPU Processing Time (with 6 cores)</td>
<td>43</td>
</tr>
<tr>
<td>No.</td>
<td>Section</td>
</tr>
<tr>
<td>-----</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>22.</td>
<td>Graphs for Example #1</td>
</tr>
<tr>
<td>23.</td>
<td>Graphs for Example #2</td>
</tr>
<tr>
<td>24.</td>
<td>Graphs for Example #3</td>
</tr>
<tr>
<td>25.</td>
<td>Graphs for Example #4</td>
</tr>
<tr>
<td>26.</td>
<td>Sioux Falls Map/Network</td>
</tr>
<tr>
<td>27.</td>
<td>Graphs for Example #5</td>
</tr>
<tr>
<td>28.</td>
<td>Graphs for Example #6</td>
</tr>
<tr>
<td>29.</td>
<td>Johnson City Map/Network</td>
</tr>
<tr>
<td>30.</td>
<td>Graphs for Example #7</td>
</tr>
<tr>
<td>31.</td>
<td>Graphs for Example #8</td>
</tr>
<tr>
<td>32.</td>
<td>Inner-Linear Projection (scaled) vs Network Size.</td>
</tr>
<tr>
<td>33.</td>
<td>Execution Using Supply to All Nodes</td>
</tr>
<tr>
<td>34.</td>
<td>Execution Using Supply to Demand Nodes Only</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

Traditional, resource consuming, algorithms employed to solve a Minimal Cost Flow (MCF) problem [ref 2] require both a Phase 1, to find the Basic Feasible Solution (BFS), and a Phase 2, to find the Optimal Final Solution (OFS). In this thesis, a simple and efficient algorithm, the Chavez-Nguyen Algorithm, (with respect to computational resources and solution time) is proposed in Phase 1 and is based on the concept of Dijkstra’s Shortest Path (SP) algorithm\(^1\); henceforth referred to as, “the Chayen algorithm”.

A warehouse distribution system analogy is a common scenario for a network-flow problem arising in Supply-Chain (or industrial) Logistics (SCL). Using a vaccine distribution analogy, we can demonstrate the significance of the capability to minimize the cost of transporting vials of vaccine from supply warehouse(s) to receiving distribution point(s); efficiencies in which not only impact business profit but may literally mean that difference between life and death. This is a typical MCF problem with warehouse/supply (source nodes) and distribution/demand (sink nodes) across a transportation [ref 3] – [ref 9] network (with intersections being transshipment nodes). The system is comprised of goods/products (in our example, vials of vaccine) to be transported along roads (arcs) across a distribution area (network) with the constraint that all the vaccine must be shipped to satisfy all the demand in a balanced (unimodal) network. The amount of vaccine available at each supply node and the amount of vaccine required at each demand node is assumed to be known.

\(^1\) An algorithm for finding the shortest paths between nodes in a graph. It was conceived by computer scientist Edsger W. Dijkstra and published in 1959.
Each arc has an associated cost, such as the gas and work required for transportation between adjacent nodes along a road, leading to a cost per vial. Minimizing the total cost per vial is the objective and can be solved by determining the cheapest route (arc paths) between supply node(s) and demand node(s) via transshipment node(s). The optimal result is the path, or a set of paths, and the associated link flows, producing the least cost per vial for transporting all of the vaccine and satisfying all of the demand in a time efficient manner.

Given the above SCL scenario, the time to generate a solution is directly proportional and dependent upon the size of the distribution area, the number of suppliers, and the number of distribution centers. Research was conducted to develop the Chayen algorithm, a mixed heuristic, SP algorithm in Phase 1 (to obtain the BFS) and the Linprgo() function in Phase 2 (to obtain the OFS), for a given network problem; a solution that is more efficient than the conventional MCF method. When compared to other, known (conventional) algorithms, the Chayen algorithm demonstrates an increase in solution generation speed (a decrease in Central Processing Unit (CPU) utilization time) while maintaining solution accuracy. Hypothesis: The Chayen algorithm will outperform both the Linprog() and conventional MCF methods with respect to timing and will also be more accurate than the conventional MCF method. While MCF problems have been well documented in many publications, due to their broad applications, little or no effort has been devoted to explaining the process of identifying the loop-formation and computing the $\theta$ value; a key value needed in calculating the link flow along loop links to solve the MCF network problems. In addition to gains in time efficiency, a detailed explanation with respect to the methodology required for computing the $\theta$ value will be presented.
A simple 5-node and 7-link (arc) network [ref 5], shown in Fig. 1, will be used to explain different steps in the conventional MCF algorithm, with special emphasis on developing an efficient algorithm for identifying the loop-formation and computing the value for $\theta$. In Fig. 1, nodes #1 and #3 are identified as supply nodes (with the values $b_1 = 6$ and $b_3 = 4$), while nodes #4 and #5 are identified as demand nodes (with the values $b_4 = -4$ and $b_5 = -5$), and node #2 is a transshipment node (with the value $b_2 = 0$). The cost ($C_{ij}$) for transporting each unit of product on any directional link i-j (from node “i” to node “j”) are also shown in Fig. 1.

![Fig. 1. 5-Node and 7-Link Network.](image)

Note: In Fig. 1, there is a supply node (#1) with six product-units available and there is a demand node (#4) that requires five product-units. A direct path from node #1 to node #4 through transshipment node #2 would cost 13 cost-units. (Not necessarily the cheapest route.)

Linear Programming (LP) is a method to achieve the best outcome in a mathematical model whose requirements are represented by linear relationships and is a special case of mathematical programming\(^2\). The MCF network presented in Fig. 1 is formulated as a LP problem [ref 10] - [ref 20], so that its special structure, one balanced equation for each node in the network (maintaining equilibrium), can be high-lighted. Details of the three steps involved in each iteration for Phase 1 and Phase 2 of the MCF algorithms are explained in Chapter II

\(^2\) [https://en.wikipedia.org/wiki/Linear_programming](https://en.wikipedia.org/wiki/Linear_programming)
Based on the conservation of flow balance (equilibrium) at each node, such that:

\[ \text{Flow Out} - \text{Flow In} = \text{Net Supply and Demand} \]  

(1)

The following equations, Eqs (2-6), can be written for the network shown in Fig. 1. (Note: “Flow Out – Flow In” accounts for the flow out and in at each node and the “Net Supply/Demand” is the total supply (positive) or demand (negative) at each node.)

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow Out – Flow In</th>
<th>Net Supply/Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+X₁₂ + X₁₃</td>
<td>= (b₁ = +6)</td>
</tr>
<tr>
<td>2</td>
<td>−X₁₂ + X₂₄ + X₂₅</td>
<td>= (b₂ = 0)</td>
</tr>
<tr>
<td>3</td>
<td>−X₁₃ + X₃₄ + X₃₅</td>
<td>= (b₃ = +4)</td>
</tr>
<tr>
<td>4</td>
<td>−X₂₄ − X₃₄ + X₄₅</td>
<td>= (b₄ = −5)</td>
</tr>
<tr>
<td>5</td>
<td>−X₂₅ − X₃₅ − X₄₅</td>
<td>= (b₅ = −5)</td>
</tr>
</tbody>
</table>

Thus, the following LP problem (Eq. 7) can be formulated from the network costs shown in Figure 1 and the balanced equations above.

The objective is to find the amount of link flow \( X_{ij} \) between nodes “i” and “j” such that the summation of the link costs times the link flows for the network is minimized:

\[
\text{Minimize COST} = \sum_{i=1}^{n} \sum_{j=1}^{n} (C_{ij} \cdot X_{ij})
\]  

(7)

with the constraints presented in Eqs. (2-6), \( n \) = number of nodes, \( C_{ij} \) = non-negative (integer) cost, and \( X_{ij} \) = non-negative (integer) flow.

It should be noted that the summation of all terms on the left-hand-side (LHS) of Eqs. (2-6) and the corresponding summation of all terms on the right-hand-side (RHS) of Eqs. (2-6)
are both equal to zero. This special (unimodal) property will allow the MCF algorithm, which can be considered a special case of LP/Simplex solver, to obtain the integer values for $X_{ij}$ at the optimum without being required to solve a more costly Linear Integer Programming (LIP) problem. From the “duality theories” [ref 10], [ref 21]-[ ref 23], [ref 24]-[ ref 27], the above 7-variable and 5-constraint “Primal” LP problem can be associated with the following 5-variable and 7-constraint “Dual” LP problem.

Find $w_1$ through $w_5$, such that $[(b_1 = 6) \cdot w_1 + (b_2 = 0) \cdot w_2 + (b_3 = 4) \cdot w_3 + (b_4 = -5) \cdot w_4 + (b_5 = -5) \cdot w_5]$ is maximized, and the following seven constraints (expressed in matrix notation) are satisfied:

\[
\begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
1 & 0 & -1 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 \\
0 & 1 & 0 & 0 & -1 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 1 & 0 & -1 \\
0 & 1 & 0 & 0 & -1
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
w_4 \\
w_5
\end{bmatrix}
= \begin{bmatrix}
C_{12} \\
C_{13} \\
C_{24} \\
C_{25} \\
C_{34} \\
C_{35} \\
C_{45}
\end{bmatrix}
\]  

(8)

The above 7x5 matrix (of the “Dual” problem) is simply the transpose of the 5x7 coefficient matrix shown in Eqs. (2-6) of the “Primal” problem. The matrix shown in Eq. (8) can also be expressed as $w_i - w_j = C_{ij}$, and this equation will be used in Chapter II.

---

3 In mathematics, unimodality means possessing a unique mode. More generally, unimodality means there is only a single optimal value, somehow defined, of some mathematical object.
The efficient algorithm for identifying the loop-formation and computing the $\theta$ value for MCF problems is also presented in Chapter II. In Chapter III, the Chayen algorithm, a mixed (hybrid) SP and the MATLAB built-in Linprog() function [ref 28] is presented and detailed.

To further verify my results, a Monte Carlo Simulation (MCS), varying the number of nodes and links (network size, number of source and sink nodes, and their connecting links), has been investigated and is elaborated upon in Chapter IV. Several academic and real-life network problems have been solved to validate the proposed algorithm. The numerical results obtained by the Chayen algorithm MATLAB$^4$ code have been compared with the built-in MATLAB Linprog() function (utilizing a Simplex algorithm) and with the conventional method (where the classical MCF algorithm is applied in both Phases 1 and 2). The have been used to validate both the accuracy and computational (time) efficiency and are presented in Chapter V. Preliminary results have indicated the Chayen algorithm has improved the solution time (without sacrificing accuracy) when compared to the conventional MCF method, which employs the MCF algorithm in both Phases 1 and 2, for the cases considered in this study. Finally, conclusions are drawn, and recommendations are made in Chapter VI.

$^4$ MATLAB version R2021a_win64 with Optimization Toolbox
CHAPTER II

THE MINIMUM COST FLOW ALGORITHM

With regards to the conventional MCF problem, the objective for Phase 1 is to find the BFS, while the objective for Phase 2 is to obtain the OFS. The MCF method is an iterative process that consists of identifying which non-basic link (not currently in the solution) should be brought into the basic (active links) set, identifying the current loop, adjusting the link flow along the loop, and removing unneeded links.

2.1 Phase 1 – Calculate the Basic Feasible Solution

After network initialization, the MCF method requires three steps to be repeated until the BFS is determined.

2.1.1 Initialization

To initialize the network and begin the iterative process of calculating the BFS, an artificial node #A (with \(b_A = 0\)) is added to the original network (shown in Fig. 1). Then, we add artificial links connected from supply nodes (node #1 and node #3) to node #A and artificial links connected from node #A to each demand and transshipment nodes (node #2, node #4, and node #5). (Note: Transshipment nodes are treated as demand nodes.) In Phase 1, all artificial links have the cost \(C_{ij} = $1\) (cost-unit), and all original links have the cost \(C_{ij} = $0\) (cost-unit), as shown in Fig. 2. This drives flow from the artificial links to the original links.
In Fig. 2, there will be 10 units of flow-IN from the supply nodes \((b_1 = 6 + b_3 = 4)\) to the artificial node #A and 10 units of flow-OUT from the artificial node #A to the demand nodes \((b_2 = 0 + b_4 = 5 + b_5 = 5)\). This network is in equilibrium; 10 units IN and 10 units OUT.

The initial cost for the network can be calculated as:

\[
\text{Min COST} = C_{1A} \cdot X_{1A} + C_{A2} \cdot X_{A2} + C_{3A} \cdot X_{3A} + C_{A4} \cdot X_{A4} + C_{A5} \cdot X_{A5}
\]

\[
= 1 \cdot (b_1 = 6) + 1 \cdot (b_2 = 0) + 1 \cdot (b_3 = 4) + 1 \cdot (b_4 = 5) + 1 \cdot (b_5 = 5)
\]

\[
= 20 \quad (9)
\]

**2.1.2 Iterative Step #1 – Calculate Node Weights**

At the beginning of each iteration, we repeatedly apply the following equation among the basic links to calculate \(w_{\#}\), with \(w_A = 0\) (associated with the artificial node #A):
\[ w_i - w_j = C_{ij} \]  

(10)

The equivalent version of Eq. (10), in matrix notation, has already been explained in Eq. (8) of Chapter I.

Then, for our 5-node example with an artificial node:

\[
\begin{align*}
    w_1 - w_A &= C_{1A}; \ w_1 - 0 = 1; \ w_1 = 1 \\
    w_A - w_2 &= C_{A2}; \ 0 - w_2 = 1; \ w_2 = -1 \\
    w_3 - w_A &= C_{3A}; \ w_3 - 0 = 1; \ w_3 = 1 \\
    w_A - w_4 &= C_{A4}; \ 0 - w_4 = 1; \ w_4 = -1 \\
    w_A - w_5 &= C_{A5}; \ 0 - w_5 = 1; \ w_5 = -1
\end{align*}
\]  

(11) \hspace{1cm} (12) \hspace{1cm} (13) \hspace{1cm} (14)

Note: All basic link costs at the beginning of the iterative process are set to 1.

Note:

a) In general, the selected #th node with the known/assigned value \( w_# \) is NOT always to be the artificial node. In the MATLAB code, \( w_# = 0 \) for node ## in which node ## has the largest number of connected basic links is selected; this should reduce the number of unknown node weights.

b) In this example, since there are a total of six nodes (five original nodes plus the artificial node), the number of basic links required to solve for all \( w_# \) is five (six nodes total minus one).

c) In applying Eq. (10), it is required that either \( w_i \) or \( w_j \) is known, so that either \( w_j \) (if \( w_i \) is known) or vice versa can be computed.

d) Since this Step #1 is straight forward, the details for the MATLAB coding are not discussed here. The MATLAB script is provided at Attachment A-1.
2.1.3 Iterative Step #2 - Determine Which Non-Basic Link Enters the Basic Set

Among the non-basic links (dashed links in Fig. 2), one needs to determine which non-basic link should be added to the basic (solid links in Fig. 2) group, based on the maximum positive value, $v$, in the following equation:

$$v = w_l - w_j - C_{ij}$$  \hspace{1cm} (17)

Then, for the current example:

- **non-basic link 1-2**  \hspace{1cm} $v = w_1 - w_2 - C_{12} = 1 - (-1) - 0 = 2$ (original cost = $8$) \hspace{1cm} (18)
- **non-basic link 1-3**  \hspace{1cm} $v = w_1 - w_3 - C_{13} = 1 - 1 - 0 = 0$ \hspace{1cm} (19)
- **non-basic link 2-4**  \hspace{1cm} $v = w_2 - w_4 - C_{24} = (-1) - (-1) - 0 = 0$ \hspace{1cm} (20)
- **non-basic link 2-5**  \hspace{1cm} $v = w_2 - w_5 - C_{25} = (-1) - (-1) - 0 = 0$ \hspace{1cm} (21)
- **non-basic link 3-4**  \hspace{1cm} $v = w_3 - w_4 - C_{34} = 1 - (-1) - 0 = 2$ (original cost = $6$) \hspace{1cm} (22)
- **non-basic link 3-5**  \hspace{1cm} $v = w_3 - w_5 - C_{35} = 1 - (-1) - 0 = 2$ (original cost = $3$) \hspace{1cm} (23)
- **non-basic link 4-5**  \hspace{1cm} $v = w_4 - w_5 - C_{45} = (-1) - (-1) - 0 = 0$ \hspace{1cm} (24)

Note:

a) Since the three non-basic links (1-2, 3-4, and 3-5) all have the same maximum positive value ($= +2$), based on Eq. (17), tie-breaker criterion needs to be considered. Although some researchers have suggested a random selection is adequate in the case of a tie, the recommended criterion is to select the link which has the minimum, original cost. Hence, in this example, link 3-5 will be selected to enter the basic group based on its original cost.

b) Since this Step #2 is straightforward, the details for MATLAB coding will not be discussed here. The MATLAB Script is provided at Attachment A-2.
2.1.4 Iterative Step #3 – Determine Which Basic Link is Removed

Among the basic links, one needs to determine which basic link(s) is/are to be removed from the basic set, by identifying the loop-formation and computing the value of $\theta$.

At the end of Step #2, it has been determined that link 3-5 should “enter the basic set”, with the flow value $X_{35} = \theta$, as shown in Fig. 3.

![Fig. 3. Link 3-5, As $\theta$, Enters the Basic Set.](image)

By “observation”, one can see that the loop can be formed by the following links 3-5 (with flow $X_{35} = \text{unknown value } \theta$), 3-A (with flow $X_{3A} = 4 - \theta$), and A-5 (with flow $X_{A5} = 5 - \theta$) as shown in Fig. 4.
The analysis is further restricted to only realistic solutions, introducing the constraint that the flow on any link must be “non-negative”, then:

\[
\begin{align*}
\text{basic link 3-5} & \quad X_{35} = \theta \geq 0 \quad (25) \\
\text{basic link 3-A} & \quad X_{3A} = 4 - \theta \geq 0; \text{hence } \theta \leq 4 \quad (26) \\
\text{basic link A-5} & \quad X_{A5} = 5 - \theta \geq 0; \text{hence } \theta \leq 5 \quad (27)
\end{align*}
\]

Based on the requirements stated in Eqs. (25-27), one concludes that the requirement \( \theta \leq 4 \) will control the outcome; when \( \theta \leq 4 \) is satisfied, \( \theta \leq 5 \) is also automatically satisfied. Thus, the maximum positive value for \( \theta \) is, \( \theta = 4 \).

Substituting the value of \( \theta = 4 \) into Eqs. (25-27), one obtains:

\[
\begin{align*}
\text{basic link 3-5} & \quad X_{35} = 4 \quad (28) \\
\text{basic link 3-A} & \quad X_{3A} = 4 - 4 = 0 \text{ (this link will be removed)} \quad (29)
\end{align*}
\]
To develop a simple algorithm which will automatically identify the loop-formation and compute the appropriate value for $\theta$, one must answer the following questions:

1: Among the basic links, how is it determined (automatically) which basic link $i$-$j$ does NOT belong to the loop-formation; which links should NOT have their flow adjusted by the value of $\theta$?

2: How is it determined (automatically) which basic links $i$-$j$ (such as basic links 3-5, 3-A, and A-5) belong to the loop-formation?

3: How is the appropriate value for $\theta$ (automatically) computed?

4: For those links that should belong to the loop-formation (such as links 3-5, 3-A, and A-5), how is it determined that the value of $\theta$ should “ADDED” or “SUBTRACTED”?

To answer the previous questions, the following terms are defined:

- A “dead node” is defined as a node which is connected by only one or no basic link (solid line), and
- A “dead basic link” is defined as a basic link (solid line) which is connected to a “dead node”; furthermore, “dead nodes” and any associated “dead links” can be DELETED (ignored or discarded) during the process of finding the $\theta$ value.

Based on the above definitions, and using Fig. 4 (focusing on basic links only), one concludes that (see Fig. 5):

- Node #1 is a “dead node” (because there is only one basic link 1-A connected to node #1), hence node #1’s associated “dead link 1-A” can be DELETED.
- Node #2 is a “dead node” (because there is only one basic link A-2 connected to node #2), hence node #2’s associated “dead link A-2” can be DELETED.
- Node #4 is a “dead node” (because there is only one basic link A-4 connected to node #4), hence node #4’s associated “dead link A-4” can be DELETED.

Fig. 5. Dead Links Deleted.

Note:

a) The remaining nodes (#3, #5, and #A), and the remaining basic links (3-A, A-5, and 3-5) will form the closed loop-formation needed to calculate the value of $\theta$.

b) All nodes that belong to the closed loop-formation will be connected (or surrounded) by at most two basic links (i.e., node #3 has link 3-5 and link 3-A; no others).
The following algorithm can be developed to automatically identify the closed loop-formation and compute $\theta$ value (in Step #3):

Step #3.1 Identify all of the “dead nodes”,

Step #3.2 Identify and DELETE all of the “dead basic links”

Step #3.3 In Step #2, it was determined that the non-basic link 3-5 will “enter the basic group”; node #3 is defined as the “loose node #i” and node #5 is defined as the “loose node #j”

Step #3.4 The “link status” for the “loose node #i = loose node #3” can be defined as one of the following four possibilities as detailed in Fig. 6:

- Link Status = 1: “out more”, (Link Status = 1 is selected see Fig. 6)
- Link Status = 2: “out less”,
- Link Status = 3: “in more”, or
- Link Status = 4: “in less”

Once the status for the current “loose node #i” is updated, the new “loose node #i” shifts to the next connected node, in this case node #A.

Step #3.5 The “link status” for the “loose node #j = loose node #5” can be defined as one of the following four possibilities as detailed in Fig. 6:

- Link Status = 1: “out more”,
- Link Status = 2: “out less”,
- Link Status = 3: “in more” (link Status = 3 is selected see Fig. 6), or
- Link Status = 4: “in less”

Once the status for the current “loose node #j” is updated, the new “loose node #j” shifts to the next connected node, in this case node #A.
Step #3.6 In Fig. 7, a basic link which is connected to the “loose node #i = loose node #3” can be EITHER case #1 OR case #4.

Fig. 6. Status of Loose Nodes #i and #j.

Fig. 7. General Case for Status Selection.
In case #1, if the basic link u-i is connected TO a loose node #i (at this point node #3), then the flow for link u-i is $X_{ui} + \theta$ (i.e., flow on u-i), so that the loose node #i will maintain its equilibrium. Under this scenario, the UPDATED loose node #i becomes loose node #u (node #A in our example), and the UPDATED link status for the loose node #u is 1, which means “out more”. Since the new flow ($= X_{ui} + \theta$) is required to be non-negative, one can have any non-negative value for $\theta$.

In case #4, if the basic link i-u is connected FROM a loose node #i (at this point node #3), then flow for link i-u is $X_{iu} - \theta$ (i.e., flow on i-u), so that the loose node #i will maintain its equilibrium. Under this scenario, the UPDATED loose node #i becomes loose node #u (node #A in our example), and the UPDATED link status for the loose node #u is 4, which means “in less”. Since the new flow ($= X_{iu} - \theta$) is required to be non-negative, the value of $\theta$ must be $\leq X_{iu}$ (LESS THAN or EQUAL TO $X_{iu}$).

Step #3.7 In Fig. 7, a basic link which is connected to the “loose node #j = loose node #5” can be EITHER case #2 OR case #3.

In case #2, if the basic link v-j is connected TO a loose node #j (at this point node #5), then the flow for link v-j is $X_{vj} - \theta$ (i.e., flow on v-j), so that the loose node #j will maintain its equilibrium. Under this scenario, the UPDATED loose node #j becomes loose node #v (node #A in our example), and the UPDATED link status for the loose node #v is 2, which means “out less”. Since the new flow ($= X_{vj} - \theta$) is required to be non-negative, the value of $\theta$ must be $\leq X_{vj}$ (LESS THAN or EQUAL TO $X_{vj}$).

In case #3, if the basic link j-v is connected FROM a loose node #j, then the flow for link j-v is $X_{jv} + \theta$ (i.e., flow on j-v), so that the loose node #j will maintain its equilibrium. Under
this scenario, the UPDATED loose node #j becomes loose node #v (node #A in our example), and the UPDATED link status for the loose node #v is 3, which means “in more”. Since the new flow \(= X_{jv} + \theta\) is required to be non-negative, one can have any non-negative value for \(\theta\).

Steps #3.6 and #3.7 are repeated until the “loose #i = loose #j” and the loop is closed.

The final value for \(\theta\) should be the smallest, positive value among cases #1, #2, #3, and #4, to guarantee the flows on every basic link of the closed loop will be non-negative and maintain equilibrium.

The MATLAB script is provided at Attachment A-3.

After Step #3 in each iteration is completed, Steps #1 through #3 will be repeated in subsequent iterations (Fig. 8 - Fig. 10), until the feasible solution, for Phase 1, is identified. The cost at the end of Phase 1 is zero because all artificial links with any flow have been removed (Fig. 10). (Note: the values displayed are the link flows \(X_{ij}\) associated with the updated \(\theta\) values or the original artificial links).
Fig. 8. End of Phase 1, Iteration 1 (Cost = 12)

Fig. 9. End of Phase 1, Iteration 2 (Cost = 10)
2.2. Phase 2 – Calculate the Optimal Final Solution (OFS)

The objective for Phase 2 is to find an overall, OFS for the original (in our example, Fig. 1, a 5-node and 7-link) network.

The BFS obtained at the end of Phase 1 (see Fig. 10) will be used as the beginning of Phase 2 (see Fig. 11). In Phase 2, the original links’ cost with the value $C_{ij}$ are used, and an #th node, with the most connected links (in this case, node #3), is selected such that $w_3 =$ known value $= 0$. Then, the same 3-step algorithm (steps #1, #2, and #3) discussed in Phase 1 will be applied for Phase2.
Fig. 11. Feasible Solution to Start Phase 2.

A complete (Phase 1 and Phase 2), step-by-step walk through of the MCF method for Chapter V Examples #1 and #2 is provided at Attachment B-1 and Attachment B-2.
CHAPTER III

THE CHAYEN ALGORITHM

As previously mentioned in Chapters I and II of this work, an MCF solution for a given network can be obtained through 2 phases. The goal for phase 1 is to find the BFS while the goal for phase 2 is to find the OFS. The conventional algorithm applies the MCF algorithm for both phases 1 and 2. In Chapter 2.1.4, a simple algorithm was proposed to identify the (closed) loop-formation, and how to implement the MATLAB code to incorporate the developed algorithm to find the $\theta$ value. The second contribution in this work is to propose a hybrid (mixed) SP algorithm [ref 3]-[ ref 4], [ref 29]-[ ref 35], the Chayen algorithm, in Phase 1 in conjunction with the Linprog() function in Phase 2. Initially I considered using the conventional MCF method for Phase 2 but that turned out to be less efficient than using the Linprog() function; this will be made clear in the analysis portion of the MCS, Chapter IV. In Fig. 12, a simple 9-node and 14-link network is used to facilitate the discussion of using the SP algorithm [ref 14] to obtain the BFS for Phase 1. The numbers associated with the links represent the cost to transport each unit of product from supply to demand nodes. Using a simple Dijkstra SP algorithm for the network in Fig. 12, an $(N+1) \times (M+1)$ matrix will be constructed. In this matrix, $N=$number of (rows) demand nodes ($= 2$, nodes #1 and #9), and $M=$number of (columns) supply nodes ($= 2$, nodes #3 and #7) and results in the $3 \times 3$ Rank/Path Matrix (this is the initialization step in the Chayen algorithm) displayed in Table 1:
Fig. 12. Simple 9-Node and 14-Link Network.

<table>
<thead>
<tr>
<th>Rank/Path Matrix</th>
<th>From Supply Node 3</th>
<th>From Supply Node 7</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Supply</td>
<td>Cost = 12; Path {3-5, 5-1}</td>
<td>Cost = 8; Path {7-5, 5-1}</td>
<td>Supply = 5</td>
</tr>
<tr>
<td>To Demand Node 1</td>
<td>Cost = 10; Path {3-5, 5-9}</td>
<td>Cost = 5; Path {7-8, 8-9}</td>
<td>Supply = 3</td>
</tr>
<tr>
<td>To Demand Node 9</td>
<td>Demand = 3</td>
<td>Demand = 5</td>
<td>Supply + Demand = 16</td>
</tr>
</tbody>
</table>

Note: The values displayed are the costs \( C_{ij} \) associated with each link.
TABLE 2
INITIAL BLANK ITERATION MATRIX

<table>
<thead>
<tr>
<th>Iteration Matrix [ ]</th>
<th>From Supply Node 3</th>
<th>From Supply Node 7</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 1</td>
<td>[]</td>
<td>[]</td>
<td>3</td>
</tr>
<tr>
<td>To Demand Node 9</td>
<td>[]</td>
<td>[]</td>
<td>5</td>
</tr>
<tr>
<td>Total Supply</td>
<td>5</td>
<td>3</td>
<td>16</td>
</tr>
</tbody>
</table>

Next, select the “cheapest cost location” in the (N=2 demand nodes) x (M=2 supply nodes) matrix (don’t include totals or headers) as the pivot (supply/demand intersection) point. In this example, the cheapest cost (from supply node #7, to demand node #9) is equal to $5 (see Table 1). In the event of a tie, select the largest demand first, then the largest supply. (This is further examined in the Chayen, by-hand solution detailed in Attachment C-3.)

The cheapest cost location, from Table 1, is at the intersection of “column supply node #7” and “row demand node #9”. Associated with this cell, the supply node #7 has the total value of three product-units, while the demand node #9 has the total value five product-units. Thus, we need to ship three (the smallest of the supply or demand values) units of product to the pivot point. The updated value for supply node #7 is equal to zero ( = 3 – 3), and the updated value for demand node #9 is equal to two ( = 5 – 3). Since the updated value for the supply node #7 is equal to zero ( = 3 – 3), the remaining cells in the supply node (column) #7 are zeroed out (indicated by [0]) and the updated sum of supply = five ( = 5 + 0) plus demand = five ( = 3 + 2) is now equal to 10, as shown in Table 3:
At this point, the updated value for “Supply + Demand” is equal to 10, which is > 0, so we need to repeat the above process for Iteration 2. Select the next cheapest location in the ranking matrix (Table 1) that is associated with an empty cell in the current Iteration Matrix (Table 3). In this example, the remaining cheapest cost is equal to 10 and is located at the intersection of supply node #3 and demand node #9; this is the new pivot point.

In this case, demand node #9 has the value equal to two, while supply node #3 has the value equal to five. Thus, one needs to ship all the demand units available at node #9 (= 2) to the pivot point. Because of this decision, the updated value for the supply node #3 is now equal to three (= 5 - 2), and the updated value for the demand node #9 is now equal to zero (= 2 - 2), as shown in the Iteration Matrix [2], Table 4. The updated sum of supply = three (= 3 + 0) plus demand = three (= 3+ 0) is now equal to six.
With the updated value for “Supply + Demand” equal to six, which is > 0, we need to repeat the above process for Iteration 3. Select the next cheapest location in the Rank/Path Matrix (Table 1) that is associated with an empty cell in the current Iteration Matrix (Table 4). In this example, the remaining cheapest cost is equal to 12 and is located at the intersection of supply node #3 and demand node #1, this becomes the new pivot point.

Demand node #1 has the value equal to three, while supply node #3 also has the value equal to three. Thus, one needs to ship all the supply units available at node #3 (= 3) to the pivot point. This results in the updated value of zero (= 3 - 3) for supply node #3 and the updated value for demand node #1 is also zero (= 3 - 3), as shown in the Iteration Matrix [3], Table 5.

The updated sum of supply = zero (= 0 + 0) plus demand = zero (= 0 + 0) is now equal to zero. Phase 1 is now complete since the sum of Supply + Demand is equal to zero.

**TABLE 5**

<table>
<thead>
<tr>
<th>Iteration Matrix [3]</th>
<th>Supply Node 3</th>
<th>Supply Node 7</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demand Node 1</td>
<td>[3]</td>
<td>[0]</td>
<td>3 - 3 = 0</td>
</tr>
<tr>
<td>Demand Node 9</td>
<td>[2]</td>
<td>[3]</td>
<td>0</td>
</tr>
<tr>
<td>Total Supply</td>
<td>3 - 3 = 0</td>
<td>0</td>
<td>Supply + Demand = 0</td>
</tr>
</tbody>
</table>

With the “sum of Supply and Demand = 0”, we have reached the BFS of Phase 1, which corresponds to flows of \(X_{31} = 3; X_{39} = 2\); and \(X_{79} = 3\). From here we expand the paths, from Table 1, to calculate the expanded Basic Feasible Solution (BFS) and the feasible objective value. \(X_{31} = (X_{35} + X_{51}), X_{39} = (X_{35} + X_{59}), \text{and } X_{79} = (X_{78} + X_{89})\) (See Table 1 for the
paths). Then the feasible objective value is \((X_{35} \cdot C_{35} + X_{51} \cdot C_{51}) + (X_{35} \cdot C_{35} + X_{59} \cdot C_{59}) + (X_{78} \cdot C_{78} + X_{89} \cdot C_{89})\) (costs pulled from the original network, Fig. 12) gives us \((3 \cdot 6 + 3 \cdot 6) + (2 \cdot 6 + 2 \cdot 4) + (3 \cdot 3 + 3 \cdot 2) = 36 + 20 + 15 = 71\), as shown in Fig. 13:

![Diagram](image)

Fig. 13. Basic Feasible Solution from Phase 1.

The MATLAB output for this example can also be viewed in Chapter V, Example #1 of this work and the by-hand Chayen solution is at Attachment C-1.

Note: It should be mentioned here that if one uses the conventional MCF algorithm in Phase 1, then the obtained BFS is shown in Fig. 14 and the objective function value is 74, which is NOT as accurate (good) as the objective function value of 71, obtained using our proposed Chayen algorithm.
With the BFS obtained in Phase 1 (shown in Fig. 13) using the Chayen algorithm (rather than using the conventional MCF algorithm), we can now use this solution as the beginning input for Phase 2 and the Linprog() function to solve for the OFS. The results indicate that the BFS, derived in Phase 1, is also the OFS found using the Linprog() function for Phase 2. In this example, one was able to obtain the OFS faster using the Chayen algorithm vs the conventional MCF algorithm.

A second example, Fig. 15, introduces a more complex, but still solvable by-hand, 13-node and 20-link network. Using a simple Dijkstra SP algorithm for the network, an \((N+1) \times (M+1)\) matrix will be used as the data structure for rank, path, and iteration information. In this matrix, \(N=\text{number of (rows) demand nodes} = 3\), nodes #2, #8, and #11, and \(M=\text{number of (columns) supply nodes} = 4\), nodes #1, #4, #10, and #13) and results in the 4x5 Rank/Path Matrix displayed in Table 6:
Fig. 15. More Complex 13-Node and 20-Link Network.

### TABLE 6

**RANK/PATH MATRIX**

<table>
<thead>
<tr>
<th>Rank/Path Matrix [0]</th>
<th>From Supply Node 1</th>
<th>From Supply Node 4</th>
<th>From Supply Node 10</th>
<th>From Supply Node 13</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 2</td>
<td>Cost = 19; Path {1,4,5,10,13,2}</td>
<td>Cost = 18; Path {4,5,10,13,2}</td>
<td>Cost = 9; Path {10,13,2}</td>
<td>Cost = 5; Path {13,2}</td>
<td>Demand = 12</td>
</tr>
<tr>
<td>To Demand Node 8</td>
<td>Cost = 17; Path {1,4,5,3,8}</td>
<td>Cost = 16; Path {4,5,3,8}</td>
<td>Cost = 12; Path {10,13,8}</td>
<td>Cost = 8; Path {13,8}</td>
<td>Demand = 16</td>
</tr>
<tr>
<td>To Demand Node 11</td>
<td>Cost = 8; Path {1,12,11}</td>
<td>Cost = 15; Path {4,5,10,12,11}</td>
<td>Cost = 6; Path {10,12,11}</td>
<td>Cost = 20; Path {13,5,10,12,11}</td>
<td>Demand = 1</td>
</tr>
<tr>
<td>Total Supply</td>
<td>Supply = 9</td>
<td>Supply = 10</td>
<td>Supply = 6</td>
<td>Supply = 4</td>
<td>Supply + Demand = 58</td>
</tr>
</tbody>
</table>

From here we can start with an initial 4x5 Iteration Matrix, Table 7, where the supply/demand intersecting cells are blank:
TABLE 7
INITIAL BLANK ITERATION MATRIX

<table>
<thead>
<tr>
<th>Iteration Matrix</th>
<th>From Supply Node 1</th>
<th>From Supply Node 4</th>
<th>From Supply Node 10</th>
<th>From Supply Node 13</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>12</td>
</tr>
<tr>
<td>To Demand Node 8</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>16</td>
</tr>
<tr>
<td>To Demand Node 11</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Total Supply</td>
<td>9</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>58</td>
</tr>
</tbody>
</table>

Select the “cheapest cost location” in the (N=3 demand nodes) x (M=4 supply nodes) matrix (don’t include totals or the headers) as the pivot point (supply/demand intersection). In this example, the cheapest cost (from supply node #13, To demand node #2) is equal to $5 (reference Table 6).

The cheapest cost location is at the intersection of “column supply node #13” and “row demand node #2”. Associated with this cell, the supply node #13 has the total value of four product-units, while the demand node #2 has the total of 12 product-units. Thus, we need to ship four (smallest of the two) units of product to the pivot point. The updated value for supply node #13 is equal to zero (= 4 – 4), and the updated value for demand node #2 is equal to eight (= 12 – 4). Since the updated value for the supply node #13 is equal to zero (= 3 – 3), the remaining cells in the supply node (column) #13 are zeroed out (indicated by [0]). The updated sum of supply = 25 (= 9 + 10 + 6 + 0) plus demand = 25 (= 8 + 16 + 1) is now equal to 50, as shown in Table 8:
The updated value for “Supply + Demand” is now equal to 50, which is > 0, so we need to repeat the above process for Iteration 2. Select the next cheapest location in the Rank/Path Matrix (Table 6) that is associated with an empty cell in the current Iteration Matrix (Table 6). In this example, the remaining cheapest cost is equal to six, and is located at the intersection of supply node #10 and demand node #11; this is the new pivot point.

In this case, demand node #11 has the value equal to one, while supply node #10 has the value equal to six. Thus, one needs to ship all the demand units available at node #11 ( = 1) to the pivot point. Because of this decision, the updated value for the supply node #10 is now equal to five ( = 6 - 1), and the updated value for the demand node #11 is now equal to zero ( = 1 - 1). Since the updated value for the demand node #11 is equal to zero ( = 1 – 1), the remaining cells in the demand node (row) #11 are zeroed out (indicated by [0]). The updated sum of supply = 24 ( = 9 + 10 + 5 + 0) plus demand = 24 ( = 8 + 16 + 0) is now equal to 48 as shown in the Iteration Matrix [2], Table 9.

**TABLE 8**

**ITERATION MATRIX[1]**

<table>
<thead>
<tr>
<th>Iteration Matrix</th>
<th>From Supply Node 1</th>
<th>From Supply Node 4</th>
<th>From Supply Node 10</th>
<th>From Supply Node 13</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ 4 ]</td>
<td>12 – 4 = 8</td>
</tr>
<tr>
<td>To Demand Node 8</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ 0 ]</td>
<td>16</td>
</tr>
<tr>
<td>To Demand Node 11</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ 0 ]</td>
<td>1</td>
</tr>
<tr>
<td>Total Supply</td>
<td>9</td>
<td>10</td>
<td>6</td>
<td>4 – 4 = 0</td>
<td>25 + 25 = 50</td>
</tr>
</tbody>
</table>
TABLE 9

ITERATION MATRIX [2]

<table>
<thead>
<tr>
<th>Iteration Matrix</th>
<th>From Supply Node 1</th>
<th>From Supply Node 4</th>
<th>From Supply Node 10</th>
<th>From Supply Node 13</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ 4 ]</td>
<td>8</td>
</tr>
<tr>
<td>To Demand Node 8</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ 0 ]</td>
<td>16</td>
</tr>
<tr>
<td>To Demand Node 11</td>
<td>[ 0 ]</td>
<td>[ 0 ]</td>
<td>[ 1 ]</td>
<td>[ 0 ]</td>
<td>1 – 1 = 0</td>
</tr>
<tr>
<td>Total Supply</td>
<td>9</td>
<td>10</td>
<td>6 – 1 = 5</td>
<td>0</td>
<td>24 + 24 = 48</td>
</tr>
</tbody>
</table>

The updated value for “Supply + Demand” is now equal to 48, which is > 0, so we need to repeat the above process for Iteration 3. Select the next cheapest location in the Rank/Path Matrix (Table 6) that is associated with an empty cell in the current Iteration Matrix (Table 9). In this example, the remaining cheapest cost is equal to nine and is located at the intersection of supply node #10 and demand node #2, this becomes the new pivot point.

Demand node #2 has the value equal to eight, while supply node #2 has the value equal to five. Thus, one needs to ship all the supply units available at node #10 ( = 5) to the pivot point. This results in the updated value of zero ( = 5 - 5) for supply node #10 and the updated value for demand node #2 is three ( = 8 - 5), as shown in the Iteration Matrix [3], Table 10. Since the updated value for the supply node #10 is equal to zero ( = 5 – 5), the remaining cells in the supply node (column) #10 are zeroed out (indicated by [0]). The updated sum of supply = 19 ( = 9 + 10 + 0 + 0) plus demand = 19 ( = 3 + 16 + 0) is now equal to 38 as shown in the Iteration Matrix [3], Table 10.
TABLE 10

ITERATION MATRIX[3]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ 5 ]</td>
<td>[ 4 ]</td>
<td>8 – 5 = 3</td>
</tr>
<tr>
<td>To Demand Node 8</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ 0 ]</td>
<td>[ 0 ]</td>
<td>16</td>
</tr>
<tr>
<td>To Demand Node 11</td>
<td>[ 0 ]</td>
<td>[ 0 ]</td>
<td>[ 1 ]</td>
<td>[ 0 ]</td>
<td>0</td>
</tr>
<tr>
<td>Total Supply</td>
<td>9</td>
<td>10</td>
<td>5 – 5 = 0</td>
<td>0</td>
<td>19 + 19 = 38</td>
</tr>
</tbody>
</table>

The updated value for “Supply + Demand” is now equal to 38, which is > 0, so we need to repeat the above process for Iteration 4. Select the next cheapest location in the ranking matrix (Table 6) that is associated with an empty cell in the current Iteration Matrix (Table 10). In this example, the remaining cheapest cost is equal to 16 and is located at the intersection of supply node #4 and demand node #8, this becomes the new pivot point.

Demand node #8 has the value equal to 16, while supply node #4 has the value equal to 10. Thus, one needs to ship all the supply units available at node #4 (= 10) to the pivot point. This results in the updated value of zero (= 10 - 10) for supply node #4 and the updated value for demand node #8 is six (= 16 - 10), as shown in the Iteration Matrix [4], Table 10. Since the updated value for the supply node #4 is equal to zero (= 10 - 10), the remaining cells in the supply node (column) #4 are zeroed out (indicated by [0]). The updated sum of supply = nine (= 9 + 0 + 0 + 0) plus demand = nine (= 3 + 6 + 0) is now equal to 18 as shown in the Iteration Matrix [4], Table 11.
TABLE 11

ITERATION MATRIX[4]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 2</td>
<td>[]</td>
<td>[0]</td>
<td>[5]</td>
<td>[4]</td>
<td>3</td>
</tr>
<tr>
<td>To Demand Node 8</td>
<td>[]</td>
<td>[10]</td>
<td>[0]</td>
<td>[0]</td>
<td>16 – 10 = 6</td>
</tr>
<tr>
<td>To Demand Node 11</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>Total Supply</td>
<td>9</td>
<td>10 – 10 = 0</td>
<td>0</td>
<td>0</td>
<td>9 + 9 = 18</td>
</tr>
</tbody>
</table>

The updated value for “Supply + Demand” is now equal to 18, which is > 0, so we need to repeat the above process for Iteration 5. Select the next cheapest location in the Rank/Path Matrix (Table 6) that is associated with an empty cell in the current Iteration Matrix (Table 11). In this example, the remaining cheapest cost is equal to 17 and is located at the intersection of supply node #1 and demand node #8, this becomes the new pivot point.

Demand node #8 has the value equal to six, while supply node #1 has the value equal to nine. Thus, one needs to ship all the demand units available at node #8 ( = 6) to the pivot point. This results in the updated value of zero ( = 6 - 6) for demand node #8 and the updated value for supply node #1 is three ( = 9 – 6). The updated sum of supply = three ( = 3 + 0 + 0 + 0) plus demand = three ( = 3 + 0 + 0) is now equal to six as shown in the Iteration Matrix [5], Table 12.
At the end of iteration five, the updated value for “Supply + Demand” is equal to six, which is > 0, so we need to repeat the above process for Iteration 6. Select the next cheapest location (the only remaining location) in the Rank/Path Matrix (Table 6) that is associated with an empty cell in the current Iteration Matrix (Table 12). In this example, the remaining cheapest cost is equal to 19 and is located at the intersection of supply node #1 and demand node #2, this becomes the new pivot point.

Now both the demand node #2 and supply node #1 have a value equal to three. Thus, one needs to ship all the supply units available at node #3 (= 3) to the pivot point. This results in the updated value of zero (= 3 - 3) for supply node #3 and the updated value for demand node #1 is also zero (= 3 - 3), as shown in the Iteration Matrix [6], Table 13. Phase 1 analysis is now complete since the sum of Supply + Demand is equal to zero and we have found the BFS.
TABLE 13

ITERATION MATRIX[6]

<table>
<thead>
<tr>
<th>Iteration Matrix { }</th>
<th>From Supply Node 1</th>
<th>From Supply Node 4</th>
<th>From Supply Node 10</th>
<th>From Supply Node 13</th>
<th>Total Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Demand Node 2</td>
<td>[3]</td>
<td>[0]</td>
<td>[5]</td>
<td>[4]</td>
<td>3 – 3 = 0</td>
</tr>
<tr>
<td>To Demand Node 8</td>
<td>[6]</td>
<td>[10]</td>
<td>[0]</td>
<td>[0]</td>
<td>6 – 6 = 0</td>
</tr>
<tr>
<td>To Demand Node 11</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>Total Supply</td>
<td>3 – 3 = 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 + 0 = 0</td>
</tr>
</tbody>
</table>

With the “sum of Supply and Demand = 0”, we have reached the BFS of Phase 1, which corresponds to flows of $X_{1,2} = 3; X_{1,8} = 6; X_{4,8} = 10; X_{10,2} = 5; X_{10,11} = 1; and X_{13,2} = 4$.

From here we expand the paths to calculate expanded BFS and the feasible objective value.

$X_{1,2} \Rightarrow (X_{1,4},X_{4,5},X_{5,10},X_{10,13}, \text{and } X_{13,2})$,

$X_{1,8} \Rightarrow (X_{1,4},X_{4,5},X_{5,3}, \text{and } X_{3,8})$, $X_{4,8} \Rightarrow (X_{4,5},X_{5,3}, \text{and } X_{3,8})$, $X_{10,2} \Rightarrow (X_{10,13} \text{ and } X_{13,2})$,

$X_{10,11} \Rightarrow (X_{10,12} \text{ and } X_{12,11})$, and $X_{13,2} \Rightarrow (X_{13,2})$ (See Table 6 for the paths).

Then the feasible objective value is $(X_{1,4} \ast C_{1,4} + X_{4,5} \ast C_{4,5} + X_{5,10} \ast C_{5,10} + X_{10,13} \ast C_{10,13} + X_{13,2} \ast C_{13,2}) + (X_{1,4} \ast C_{1,4} + X_{4,5} \ast C_{4,5} + X_{5,3} \ast C_{5,3} + X_{3,8} \ast C_{3,8}) + (X_{4,5} \ast C_{4,5} + X_{5,3} \ast C_{5,3} + X_{3,8} \ast C_{3,8}) + (X_{10,13} \ast C_{10,13} + X_{13,2} \ast C_{13,2}) + (X_{10,12} \ast C_{10,12} + X_{12,11} \ast C_{12,11}) + (X_{13,2} \ast C_{13,2})$ (costs pulled from the original network, Fig. 15) gives us

$(3 \ast 1 + 3 \ast 5 + 3 \ast 4 + 3 \ast 4 + 3 \ast 5) + (6 \ast 1 + 6 \ast 5 + 6 \ast 4 + 6 \ast 7) + (10 \ast 5 + 10 \ast 4 + 10 \ast 7) + (5 \ast 4 + 5 \ast 5) + (1 \ast 3 + 1 \ast 3) + (4 \ast 5) = (3 + 15 + 12 + 12 + 15) + (6 + 30 + 24 + 42) + (50 + 40 + 70) + (20 + 25) + (3 + 3) + (20) = (57) + (102) + (160) + (45) + (6) + (20) = 390$, as shown in Fig. 16.
The MATLAB output for this example can also be viewed in Chapter V, Example #2 of this thesis and the full by-hand Chayen solution is provided in Attachment C-2.

Note: It should be mentioned here that if one uses the conventional MCF algorithm in Phase 1, then the obtained BFS objective value is 470, which is significantly different from the Chayen algorithm Phase 1 solution of 390.

The BFS obtained in Phase 1 (shown in Fig. 16) using the Chayen algorithm (rather than using the conventional MCF algorithm in Phase 1) can now be used as the beginning input for Phase 2 and the Linprog() function. At the conclusion of Phase 2, we reach the optimal solution of 382 which agrees with the solution using the Linprog() function from start to finish. We can see that the Phase 1 BFS, using the Chayen algorithm, was off target by 2.1% while the Phase 1 BFS, using the conventional MCF method, was off by 23.0%.
CHAPTER IV
MONTE CARLO SIMULATION

The original hypothesis states “the Chayen algorithm will outperform both the Linprog() algorithm and the conventional MCF method with respect to timing”. To predict the timing of each method over different size networks, a Monte Carlo Simulation (MCS), using 350 randomly generated networks was conducted. Each candidate network was comprised of 8 to 57 nodes and 10 to 245 links. Before the candidate network was accepted for the MCS, it was validated using the Linprog() function to weed out any unbounded or infeasible solutions and further confirmed using the conventional MCF method to ensure no initial link was inadvertently reversed rendering the final solution invalid. (Note: As the network size grew, the less likely it was that the MCF method would be able to solve the problem without some form of link reversal due to a degenerate network. The issue of degeneration is not addressed in this work but is recommended as a future research topic.)

The solution generation time of each of the 350 networks were timed using the Linprog() function, the Chayen algorithm using the Linprog() function in Phase 2, the Chayen algorithm using the conventional MCF method in Phase 2, and the conventional MCF method (in both Phase 1 and Phase 2); this constituted one trial. There were 36 trials conducted, and the median 30 results (removing outliers) were averaged to generate the recorded time for each method of each network. Fig. 17 shows the number of Phase 1 iterations required to solve the problem for both the Chayen algorithm and the conventional MCF method; it is clear that, as the size of the network grows, the number of iterations also increases. However, the rate of increase in

---

5 A MCS is a computational algorithm that relies on random sampling to obtain numerical results.
iterations using the Chayen algorithm is consistently smaller than the rate of increase using the conventional MCF method. It was noted that there was a general relationship between the number of supply nodes and demand nodes to the number of iterations in the Chayen algorithm—the iterations roughly equated to the number of supply nodes plus the number of demand nodes minus one. For example, if the network contains four supply nodes and six demand nodes, it generally meant that the Chayen, Phase 1, algorithm required nine (\(= 4 + 6 - 1\)) iterations. No correlation between supply and demand nodes was observed for the conventional MCF method.

The overall MCS results backed the original hypothesis that “the Chayen algorithm will outperform both the Linprog() and conventional MCF methods with respect to timing” for this set of 350 random networks (see Fig. 18). Further, it was demonstrated that 95.7% of the time, the Chayen algorithm produced the optimal solution at the end of Phase 1 compared to 74.4% of the time for the conventional MCF method. On top of that, the overall accuracy at the end of Phase 1 was off, on average, by only 4.3% for the Chayen algorithm vs 41.1%, on average, for the conventional MCF method which, ultimately, reduces the amount of effort required for the Chayen algorithm to reach the OFS at the end of Phase 2. This supported the second part of the hypothesis, “The Chayen algorithm will also be more accurate than the conventional MCF method.”

The simulation was executed using MATLAB on a Dell i7-8700 CPU at 3.2GHz with 32MB RAM and 6 core processors. The data used in this Monte Carlo simulation is provided at Attachment D.
Fig. 17. Solution Iterations vs Network Size.
Fig. 18. Processing Time vs Network Size.
Using an inner-linear interpolation and MS Excel’s FORECAST function as well as MS Excel’s exponential projection for each method, one was able to scale the results from Fig. 18 to present a better look at the results for the 350 MCS networks (Fig. 19). Notice that, not only does the Chayen+MCF algorithm (Chayen in Phase 1 and conventional MCF in Phase 2) consumes more execution time than the Linprog() function. It also appears that it will eventually consume more execution time than the conventional MCF method. This phenomenon can be contributed to the increasing network size requiring the MCF method to spend more time in Step 1, finding node weights, and in a portion of Step 3, finding the loop amongst an ever-increasing set of links. Also, the time required in the Chayen algorithm to calculate the shortest path increases as the network size grows.

Fig. 19. 350 Network Execution Time vs Size (scaled)
In addition to comparing the execution time (wall-clock time) for each network and method using MATLAB’s built-in “tic”/”toc” functions, the CPU processor time for each network and method, using MATLAB’s built-in “cputime” function, has also been considered. However, this proved inconclusive as the “cputime” function measures CPU processing time across ALL cores and then sums them together. By default, MATLAB’s built-in functions are highly optimized and will automatically take advantage of multiple processors as deemed necessary (whether requested or not) the results are not comparable across the analysis methods. This can be seen in the comparison between Fig. 20 and Fig. 21. Of note, the Chayen algorithm seems to remain consistent across both the Wall-Clock time measurements and the CPU Processing time measurements. Whereas the MATLAB built-in Linprog() function reports a greater CPU Processing time which is indicative of Linprog() being executed across multiple CPUs simultaneously.
Fig. 20. Wall-Clock Execution Time

Fig. 21. CPU Processing Time (with 6 cores)
Based on the 3-step MCF algorithm (applied in Phase 1 and Phase 2) presented in Chapter II, and the newly proposed shortest path (SP) algorithm used in Phase 1 and presented in Chapter III, the following academic and real-life networks [ref 36] are used to validate our proposed algorithm for identifying the closed loop-formation and computing the appropriate value for $\theta$ (see the discussions in Chapter 2.1.4, for Step #3 of Phase 1 and Phase 2). All our optimum results, for these examples, have been compared with, and validated using, the built-in MATLAB Linprog() function (Simplex algorithm) and the results were recorded with each example in this chapter. The process times were measured over 36 trials and the average of the median 30 measurements were included in the analysis of each example.
Example #1: 9-Node/14-Link

Fig. 22a. Original Network (9/14)

Fig. 22b. Linprog()
Solution (9/14) = 71

Fig. 22c. Chayen Phase 1
Solution (9/14) = 71

Fig. 22d. Chayen Phase 2
Solution (9/14) = 71

Fig. 22e. Conventional Phase 1
Solution (9/14) = 80

Fig. 22f. Conventional Phase 2
Solution (9/14) = 71

Fig. 22. Graphs for Example #1
TABLE 14

EXAMPLE #1 - Timing and Iteration Results

<table>
<thead>
<tr>
<th></th>
<th>Linprog</th>
<th>Chayen</th>
<th>MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time (ms)</td>
<td>94.3</td>
<td>20.5</td>
<td>23.2</td>
</tr>
<tr>
<td>Phase 1 Iterations</td>
<td></td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Phase 2 Iterations</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

The Chayen algorithm provides the optimal solution at the end of Phase 1 (Fig. 22c) while the conventional MCF method required Phase 2 to refine the feasible solution and deliver the optimal solution (Fig. 22f). Both the Chayen and conventional MCF methods produce the same flow as the Linprog() solution (Fig. 22b). The MCF solution required 2.3 times the number of iterations required for the Chayen algorithm in Phase 1. The Chayen algorithm was on target at the end of Phase 1 while the conventional MCF method missed the target, at the end of Phase 1, by 12.7%

The MCF, by-hand, solution can be found at Attachment B-1 and the Chayen, by-hand, solution can be found at Attachment C-1.
Example #2: 13-Node/20-Link

Fig. 23a. Original Network (13/20)

Fig. 23b. Linprog()
Solution (13/20) = 382

Fig. 23c. Chayen Phase 1
Solution (13/20) = 390

Fig. 23d. Chayen Phase 2
Solution (13/20) = 382

Fig. 23e. Conventional Phase 1
Solution (13/20) = 470

Fig. 23f. Conventional Phase 2
Solution (13/20) = 382

Fig. 23. Graphs for Example #2
TABLE 15

EXAMPLE #2 - Timing and Iteration Results

<table>
<thead>
<tr>
<th></th>
<th>Linprog</th>
<th>Chayen</th>
<th>MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time (ms)</td>
<td>26.6</td>
<td>15.4</td>
<td>36.9</td>
</tr>
<tr>
<td>Phase 1 Iterations</td>
<td>6</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Phase 2 Iterations</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

In this example, both the Chayen algorithm and the conventional MCF method required a Phase 2 to find the optimal solution. Notice that the Chayen algorithm produces a better feasible solution from Phase 1 (Fig. 23c) than the MCF method (Fig. 23e.). Both the Chayen and conventional MCF methods produce the same flow and final, OFS as the Linprog() solution (Fig. 23b). The MCF solution required 1.9 times the number of iterations required for the Chayen algorithm in Phase 1. The Chayen algorithm was off target at the end of Phase 1 by 2.1% while the conventional MCF method missed the target, at the end of Phase 1, by 23.0%.

The MCF, by-hand, solution can be found at Attachment B-2 and the Chayen, by-hand, solution can be found at Attachment C-2.
Example #3: 20-Node/74-Link

Fig. 24a. Original Network (20/74)

Fig. 24b. Linprog()
Solution (20/74) = 71

Fig. 24c. Chayen Phase 1
Solution (20/74) = 75

Fig. 24d. Chayen Phase 2
Solution (20/74) = 71

Fig. 24e. Conventional Phase 1
Solution (20/74) = 71

Fig. 24f. Conventional Phase 2
Solution (20/74) = 71

Fig. 24. Graphs for Example #3
TABLE 16

EXAMPLE #3 - Timing and Iteration Results

<table>
<thead>
<tr>
<th></th>
<th>Linprog</th>
<th>Chayen</th>
<th>MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time (ms)</td>
<td>38.2</td>
<td>29.9</td>
<td>88.1</td>
</tr>
<tr>
<td>Phase 1 Iterations</td>
<td>9</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Phase 2 Iterations</td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

In this example, both the Chayen algorithm and the conventional MCF method required a Phase 2 to find the OFS. Both the Chayen and conventional MCF methods produce the same OFS as the Linprog() solution (Fig. 24b); however, both the Chayen algorithm and the conventional MCF method produces an alternate optimal flow (see Fig. 24d and Fig. 24f). This indicates that there is more than one optimal path that results in the same cost. In the case of vaccine distribution, this is saying that it will cost the same amount whether you follow the Linprog() produced path, the Chayen algorithm produced path, or the conventional MCF produced path. The MCF solution required 1.8 times the number of iterations required for the Chayen algorithm in Phase 1.

In this rare example, the conventional MCF method produced the OFS at the end of Phase 1 while the Chayen algorithm, at the end of Phase 1, was off target by 5.6%.
Example #4: 24-Node/76-Link - Sioux Falls

Fig. 25a. Original Network (24/76)

Fig. 25b. Linprog()
Solution (24/76) = 83

Fig. 25c. Chayen Phase 1
Solution (24/76) = 83

Fig. 25d. Chayen Phase 2
Solution (24/76) = 83

Fig. 25e. Conventional Phase 1
Solution (24/76) = 188

Fig. 25f. Conventional Phase 2
Solution (24/76) = 83

Fig. 25. Graphs for Example #4
TABLE 17

EXAMPLE #4 - Timing and Iteration Results

<table>
<thead>
<tr>
<th></th>
<th>Linprog</th>
<th>Chayen</th>
<th>MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing Time (ms)</strong></td>
<td>38.6</td>
<td>29.2</td>
<td>74.1</td>
</tr>
<tr>
<td><strong>Phase 1 Iterations</strong></td>
<td>6</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td><strong>Phase 2 Iterations</strong></td>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

In this real-life, Sioux Falls example (Fig. 26), the Chayen algorithm provides the optimal solution value at the end of Phase 1 (Fig. 25c) while the conventional MCF method required Phase 2 to refine the feasible solution and deliver the OFS (Fig. 25e). Both the Chayen and conventional MCF methods produce the same OFS as the Linprog() solution (Fig. 25b); however, both the Chayen algorithm and the conventional MCF method produces the same alternate optimal flow; indicating that there is more than one path that will result in the same optimal value or cost. The conventional MCF solution required 3.1 times the number of iterations required for the Chayen algorithm in Phase 1. At the end of Phase 1, the conventional MCF method missed the target by 58.3%.

Fig. 26. Sioux Falls Map/Network
Example #5: 33-Node/104-Link

Fig. 27a. Original Network (33/104)

Fig. 27b. Linprog()
Solution (33/104) = 433

Fig. 27c. Chayen Phase 1
Solution (33/104) = 445

Fig. 27d. Chayen Phase 2
Solution (33/104) = 433

Fig. 27e. Conventional Phase 1
Solution (33/104) = 589

Fig. 27f. Conventional Phase 2
Solution (33/104) = 433

Fig. 27. Graphs for Example #5
In this example, both the Chayen algorithm and the conventional MCF method required a Phase 2 to find the optimal solution. Notice that the Chayen algorithm produces a better feasible solution from Phase 1 (Fig. 27c) than the conventional MCF method (Fig. 27e). Both the Chayen and conventional MCF methods produced the same flow and OFS as the Linprog() solution (Fig. 27b). The conventional MCF solution required 1.9 times the number of iterations required for the Chayen algorithm in Phase 1. The Chayen algorithm was off target at the end of Phase 1 by 2.8% while the conventional MCF method missed the target, at the end of Phase 1, by 36.0%.

The Chayen Phase 1, by-hand, solution can be found at Attachment C-3.
Example #6: 50-Node/102-Link – Johnson City

Fig. 28a. Original Network (50/102)

Fig. 28b. Linprog()
Solution (50/102) = 56000

Fig. 28c. Chayen Phase 1
Solution (50/102) = 56000

Fig. 28d. Chayen Phase 2
Solution (50/102) = 56000

Fig. 28e. Conventional Phase 1
Solution (50/102) = 68000

Fig. 28f. Conventional Phase 2
Solution (50/102) = 68000

Fig. 28. Graphs for Example #6
TABLE 19
EXAMPLE #6 - Timing and Iteration Results

<table>
<thead>
<tr>
<th></th>
<th>Linprog</th>
<th>Chayen</th>
<th>MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time (ms)</td>
<td>45.0</td>
<td>36.2</td>
<td>384.3</td>
</tr>
<tr>
<td>Phase 1 Iterations</td>
<td>12</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Phase 2 Iterations</td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

In this real-life, Johnson City example (Fig. 29), the Chayen algorithm provides the OFS value at the end of Phase 1 (Fig. 28c). Like the Chayen algorithm, the conventional method reaches its final solution at the end of Phase 1 but does not produce an optimal solution value. In this case, the network is a degenerate network and forces an original, one-way link to reverse direction (link 26→29 is reversed and is now 29→26). This would be as if the vaccine delivery, following the provided path, ran into a one-way road going the wrong direction; delivery failure. The concept of degeneration should be address with future research (see Chapter VI).

This network was not included in the overall timing measurements because the optimal solution from the conventional MCF method did not agree with the Linprog() solution.

Fig. 29. Johnson City Map/Network
Example #7: 225-Node/375-Link

Fig. 30a. Original Network (225/375)

Fig. 30b. Linprog()
Solution (225/375) = 1775

Fig. 30c. Chayen Phase 1
Solution (225/375) = 1775

Fig. 30d. Chayen Phase 2
Solution (225/375) = 1775

Fig. 30e. Conventional Phase 1
Solution (225/375) = 2000

Fig. 30f. Conventional Phase 2
Solution (225/375) = 1775

Fig. 30. Graphs for Example #7
TABLE 20
EXAMPLE #7 - Timing and Iteration Results

<table>
<thead>
<tr>
<th></th>
<th>Linprog</th>
<th>Chayen</th>
<th>MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time</td>
<td>112.6</td>
<td>212.9</td>
<td>1,518.2</td>
</tr>
<tr>
<td>Phase 1 Iterations</td>
<td></td>
<td>75</td>
<td>151</td>
</tr>
<tr>
<td>Phase 2 Iterations</td>
<td></td>
<td></td>
<td>26</td>
</tr>
</tbody>
</table>

The Chayen algorithm provides the optimal solution at the end of Phase 1 (Fig. 30c) while the conventional MCF method required Phase 2 to refine the feasible solution and deliver the optimal solution (Fig. 30f). Both the Chayen and conventional MCF methods produce the same flow as the Linprog() solution (Fig. 30b). The conventional MCF solution required 2.3 times the number of iterations required for the Chayen algorithm in Phase 1. At the end of Phase 1, the conventional MCF method missed the target by 12.7%. This network was artificially generated by replicating a solved 9-node and 14-link network 25 times in order to capture timing measurements for larger networks.
Example #8: 900-Node/1500-Link

Fig. 31a. Subsection of Original Network (900/1500)

Fig. 31b. Linprog() Solution (900/1500) = 7100

Fig. 31c. Chayen Phase 1 Solution (900/1500) = 7100

Fig. 31d. Chayen Phase 2 Solution (900/1500) = 7100

Fig. 31e. Conventional Phase 1 Solution (900/1500) = 8000

Fig. 31f. Conventional Phase 2 Solution (900/1500) = 7100

Fig. 31. Graphs for Example #8
TABLE 21

EXAMPLE #8 - Timing and Iteration Results

<table>
<thead>
<tr>
<th></th>
<th>Linprog</th>
<th>Chayen</th>
<th>MCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time (ms)</td>
<td>408.6</td>
<td>2,479.4</td>
<td>1,770,581.0</td>
</tr>
<tr>
<td>Phase 1 Iterations</td>
<td>300</td>
<td>601</td>
<td></td>
</tr>
<tr>
<td>Phase 2 Iterations</td>
<td>1</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>

The Chayen algorithm provides the optimal solution at the end of Phase 1 (Fig. 31c) while the conventional MCF method required Phase 2 to refine the feasible solution and deliver the optimal solution (Fig. 31f). Both the Chayen and conventional MCF methods produce the same flow as the Linprog() solution (Fig. 31b). The conventional MCF solution required 2.3 times the number of iterations required for the Chayen algorithm in Phase 1. At the end of Phase 1, the conventional MCF method missed the target by 12.7%. This network was artificially generated by replicating a solved 9-node and 14-link network 100 times in order to capture timing measurements for larger networks.
CHAPTER VI

CONCLUSION AND FUTURE CONSIDERATIONS

The most complicated step in the conventional MCF (Phase 1 and Phase 2) algorithm occurs in Step #3 (identifying the closed loop and computing the appropriate value for $\theta$).

Searching for and reviewing material related to the MCF method revealed no existing literature that clearly discusses the (automated) numerical algorithm to accomplish Step #3 of the conventional MCF method. This work has explained, in substantial detail, the step-by-step procedures to implement the conventional MCF algorithm, with emphasis on Step #3 (see Chapter 2.1.4) to determine the loop-formation and value for $\theta$ (complete MCF solutions, by-hand, for Chapter V examples #1 and #2 can be found at Attachment B-1 and Attachment B-2).

Furthermore, this work has also proposed a new heuristic; hybrid SP Chavez-Nguyen Algorithm (in Phase 1) in conjunction with the Linprog() function (in Phase 2), rather than applying the conventional MCF method in both Phases 1 and 2, to reduce computational time in networks of increasing size and complexity (complete MCF solutions, by-hand, for Chapter V examples #1 and #2 can be found at Attachment C-1 and Attachment C-2).

Originally, the conventional MCF method was considered for Phase 2 of the Chayen algorithm, but analysis revealed that this approach would require additional time when resolving the node weight calculations in Step 1 of the three-step process. After collecting timing information for 20 random networks, it is even more prominent in situations that are highly degenerate. The data in Table 22 shows the network size, the percent of disconnected nodes at the beginning of Phase 2 (for the degenerate cases), and the time required by Step 1. If we consider the network size as the number of nodes multiplied by the number of links (total...
combinations) we observe networks number 6 and 7 have roughly the same size. Continuing this examination, it is clear that network 7 requires a significant amount of work to resolve the weights with 73% disconnected nodes in Step 1 vs only 21% disconnected nodes for network 6. Step 1 for network 7 requires 4.8 times more time than network 6. (This phenomenon also occurs between networks 3 and 4.) So, it is theorized that, not only does Phase 2 require more time based on the size of the network but the overall time is also directly affected by the number of disconnected nodes contained in the Phase 1 BFS. Therefore, it was concluded that the use of the Linprog() function in Phase 2 for the Chayen algorithm provided better timing results by removing this phenomenon from the overall Chayen algorithm.

### TABLE 22

Phase 2, Step 1 Processing Times

<table>
<thead>
<tr>
<th>Network</th>
<th>Nodes</th>
<th>Links</th>
<th>Size</th>
<th>% Disconnected Links</th>
<th>Step 1 Time based on average of 3 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>12</td>
<td>108</td>
<td>67%</td>
<td>0.0270</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>14</td>
<td>126</td>
<td>33%</td>
<td>0.0220</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>25</td>
<td>250</td>
<td>80%</td>
<td>0.0343</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>20</td>
<td>260</td>
<td>23%</td>
<td>0.0227</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>35</td>
<td>280</td>
<td>38%</td>
<td>0.0263</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>51</td>
<td>714</td>
<td>21%</td>
<td><strong>0.0323</strong></td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>48</td>
<td>720</td>
<td>73%</td>
<td><strong>0.1570</strong></td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>74</td>
<td>1480</td>
<td>40%</td>
<td>0.1243</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>52</td>
<td>1560</td>
<td>30%</td>
<td>0.0390</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>59</td>
<td>1770</td>
<td>87%</td>
<td>0.0487</td>
</tr>
<tr>
<td>11</td>
<td>25</td>
<td>74</td>
<td>1850</td>
<td>56%</td>
<td>0.0353</td>
</tr>
<tr>
<td>12</td>
<td>23</td>
<td>96</td>
<td>2208</td>
<td>70%</td>
<td>0.0380</td>
</tr>
<tr>
<td>13</td>
<td>25</td>
<td>105</td>
<td>2625</td>
<td>56%</td>
<td>0.0403</td>
</tr>
<tr>
<td>14</td>
<td>33</td>
<td>104</td>
<td>3432</td>
<td>24%</td>
<td>0.1253</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>137</td>
<td>4110</td>
<td>70%</td>
<td>0.0493</td>
</tr>
</tbody>
</table>
Table 22 (continued)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>47</td>
<td>91</td>
<td>4277</td>
<td>94%</td>
</tr>
<tr>
<td>17</td>
<td>43</td>
<td>108</td>
<td>4644</td>
<td>91%</td>
</tr>
<tr>
<td>18</td>
<td>50</td>
<td>102</td>
<td>5100</td>
<td>40%</td>
</tr>
<tr>
<td>19</td>
<td>225</td>
<td>375</td>
<td>84375</td>
<td>33%</td>
</tr>
<tr>
<td>20</td>
<td>900</td>
<td>1500</td>
<td>1350000</td>
<td>33%</td>
</tr>
</tbody>
</table>

Again, using an inner-linear interpolation and MS Excel’s FORECAST function, the timings from 350 networks used for the MCS in Chapter IV along with the addition of three larger networks (225-nodes x 375-links, 416-nodes x 914-links, and 900-nodes x 1500-links), the overall scaled process timing has been predicted for the networks examined in this study. The results are displayed in Fig. 32 and partially support the original hypothesis that the Chayen algorithm will outperform both the Linprog() and conventional MCF methods with respect to timing in networks exceeding a size of approximately 135 nodes and 135 links. It is around this point that the Chayen algorithm exceeds the time required by the Linprog() algorithm. It is theorized that the efficiency of the code, for the Chayen algorithm, could be optimized to eventually remain faster than the Linprog() function taking advantage of the near optimal solution produced by Phase 1. An initial attempt to optimize the SP algorithm was evaluated. The code was modified FROM specifically calculating the shortest path from each supply node to all demand nodes through all other nodes TO using MATLAB’s built-in Shortestpath() function (hoping to take advantage of the highly optimized function) is shown in the comparison between Fig. 33 and Fig. 34.
Fig. 32. Inner-Linear Projection (scaled) vs Network Size.
Fig. 33. Execution Using Supply to All Nodes

Fig. 34. Execution Using Supply to Demand Nodes Only
In the above comparisons (Fig. 33 and Fig. 34) the initial gap between the Chayen and Linprog() timing is the same at 8.8 ms but the ending gap, at approximately 116-nodes x 116-links is 7.5 ms for the Supply-to-All and 8.3 ms for the Supply-to-Demand only. This is equivalent to a 14.7% reduction vs a 5.7% reduction indicating that the gap using the Supply-to-All is closing faster than the gap using Supply-to-Demand. So there is a gain, albeit minimal, using the built-in Shortestpath() function, but still not enough to prevent the Chayen algorithm from eventually crossing over the Linprog() function.

This directly leads to an opportunity for additional research. First, to increase the size of the networks that the conventional MCF method can solve, it is imperative that the link reversal issue, associated with degenerated networks, be resolved. The link reversal occurs when a much earlier iteration resulted in a degenerated situation and required more than one non-basic link be added to the basic group to compute the required node, \( w_x \), weights (this is step 1 of the conventional MCF method). Then, later, one of those links that was brought in, is forced to change direction. While this shouldn’t affect the results if the link is bi-directional, in the cases where the link is one-way and forced to change direction the solution is rendered invalid. (see Chapter V, Example #6.) As larger networks are able to be analyzed, the overall results will improve and may lead to a better understanding and identification of areas of improvement in efficiency.

Second, another opportunity for an improvement in processing time is the exploration of parallelism in the analysis and generation of the shortest paths required in the Chayen algorithm for much larger networks. Because each path and cost, in the initialization step of the Chayen algorithm, can be independently calculated, the concept of modifying this serial process to take
advantage of a parallel architecture could result in a significant time savings as the size of the network increases. Initial observations using four processors, for networks of small to medium size, seem to indicate that the parallel setup overhead and cross-processor communication is actually detrimental to the execution timing; however, theoretically the communications and setup overhead may have less of an impact when analyzing large networks and this may be worth exploring.

Thirdly, expanding on the parallelism mentioned above, pursuing the concept of splitting a large network into many smaller networks and then recombining the results could prove beneficial. This could take advantage of the reliably faster speed and accuracy the Chayen algorithm exhibits processing networks below a certain size threshold. And each smaller network could be processed across multiple CPUs thereby further reducing the processing time for the larger network. It is theorized that the smaller networks and their optimal, local solutions may be able to be recombined and used as entry criteria into a Phase 3 which may produce the OSF for the entire network in a way that may eventually save more processing time.

And finally, it may be worth the effort to explore the utility of the Amazon Web Services (AWS) and Microsoft’s Azure Artificial Intelligence (AI) / Machine Learning (ML) systems as a way of optimizing the current algorithm and addressing the degenerated network situation. ML may introduce an “intuitiveness” to solving large networks without introducing link reversals leading to an invalid solution.

This thesis examined a new method for solving SCL challenges, with significant real-world implications on cost, time, and impact. Our analysis showed that the proposed Chayen algorithm to be superior (with respect to accuracy) to the current conventional MCF method for any network size. For small-to-moderate network sizes (i.e., less than 135-nodes and 135-links),
our proposed Chayen algorithm has outperformed (with respect to solution generation time and accuracy) the built-in MATLAB Linprog() function throughout.
REFERENCES


ATTACHMENTS

ATTACHMENT A – MATLAB SCRIPTS

MATLAB is a platform developed by Mathworks, Inc. and is used to analyze data, develop algorithms, and create models. The MATLAB Scripts for this work were developed using MATLAB version 9.10.0.1538726 (R2021a) Prerelease.
A.1. Conventional MCF Method, Step #1

function [w, basic_links, curr_orig_link_cost, node_i, node_j, nonbasic_links, link_cost, link_flows, orig_link_cost, link_values] = F_MCFStep1(phase, link_cost, num_basic_links, num_nodes, basic_links, curr_orig_link_cost, node_i, node_j, nonbasic_links, num_nonbasic_links, link_flows, orig_link_cost, link_values, outFileID)

% initialize values for array wi (node weights) to -9999, and % initialize node <known_node_wi>, with most basic links, to zero
if(phase == 1)
    % Phase 1 includes the artificial node and links
    w = (zeros(1,(num_nodes + 1)) + 1) * -9999;
    countLinks = zeros(1,(num_nodes+1));
else % Phase 2 - does not include the artificial node and links
    w = (zeros(1,(num_nodes)) + 1) * -9999;
    countLinks = zeros(1,(num_nodes));
end

for i = 1:(num_basic_links) % count basic links attached to each node
    indx = (num_nonbasic_links + i);
    countLinks(node_i(indx)) = countLinks(node_i(indx)) + 1;
    countLinks(node_j(indx)) = countLinks(node_j(indx)) + 1;
end

 [~,keyNode] = max(countLinks); % first node with most links
w(keyNode) = 0; % node with most basic links is set to zero
fprintf(outFileID,'Setting w%d=0
',keyNode);
% include all non-basic links to identify any orphaned nodes
for i = 1:(num_nonbasic_links) % add nonbasic links
    indx = i;
    countLinks(node_i(indx)) = countLinks(node_i(indx)) + 1;
    countLinks(node_j(indx)) = countLinks(node_j(indx)) + 1;
end
ListOfNodes = (1:size(w,2)); % initialize the list of nodes
OrphanedNodes = ListOfNodes(countLinks == 0); % identify orphans
if(sum(OrphanedNodes) > 0) % mark orphaned nodes
    w(OrphanedNodes) = -9998;
end
num_iterations = num_nodes; % arbitrarily selecting # of iterations
Done = false;
Found = false;
while(~Done) % It will only be Done when all w(i) are calculated
    [w] ... = F_Sub1MCFStep1(link_cost, node_i, node_j, num_basic_links, ...
    num_iterations, num_nonbasic_links, w);
    unknown_w_nodes = ListOfNodes(w == -9999); % initialize unknown
    % Make sure at least one end of the link is connected to a node
    % within the set of known nodes
    known_w_nodes = ListOfNodes(w ~= -9999 & w ~= -9998);
    for x1 = size(unknown_w_nodes,2):-1:1 % travers in reverse order
        opposites = [];
        for x2 = 1:num_nonbasic_links %#522 can use parfor: NO
            if (node_i(x2) == unknown_w_nodes(x1))
                opposites = [opposites node_j(x2)];
            end
            if (node_j(x2) == unknown_w_nodes(x1))
                opposites = [opposites node_i(x2)];
            end
        end
        Found = false;
        for x3 = 1:size(opposites,2)
            if(any(known_w_nodes == opposites(x3)))
                Found = true;
                break;
            end
        end
        if(~Found)
            unknown_w_nodes(x1) = [];
        end
    end
    extra_basic_links = []; % clear the extra links array
    % display weights - status information
    if(isempty(unknown_w_nodes))
        fprintf(outFileID,'w(i) =     [ ');
        for indx = 1:size(w,2)
            fprintf(outFileID,'%3d',w(indx));
        end
        fprintf(outFileID,'
');
        Done = true;
    else
        % identify unknown node with the most links
        fprintf(outFileID,'Need extra basic link to solve for w\n');
        [~,nodeToPullLink] = max(countLinks(unknown_w_nodes));
        node_unknown = unknown_w_nodes(nodeToPullLink);
    end
end
count_extra_basic_link = 0;
for link = 1:1:num_nonbasic_links
    inode = node_i(link);
    jnode = node_j(link);
    if (node_unknown == inode || node_unknown == jnode)
        count_extra_basic_link = count_extra_basic_link + 1;
        extra_basic_links(count_extra_basic_link) = link;
    end
end
if (count_extra_basic_link == 0) % this should not occur
    fprintf(outFileID,'   Error : Could not get non-basic');
    fprintf(outFileID,' link for node: %d
',node_unknown);
    w(node_unknown) = -9998; % not needed
else
    link_to_add = 0;
    if (count_extra_basic_link > 1)
        if(phase == 1) % Phase 1 link selection
            % Original method Cheapest OUTGOING First (Phase 1)
            outgoing = 0; outgoing_links = [];
            incoming = 0; incoming_links = [];
            for x = 1:1:count_extra_basic_link % sort links
                link = extra_basic_links(x);
                inode = node_i(link);
                jnode = node_j(link);
                if (inode == node_unknown)% as an outgoing link
                    outgoing = outgoing + 1;
                    outgoing_links(outgoing) = link;
                else % or an incoming link
                    incoming = incoming + 1;
                    incoming_links(incoming) = link;
                end
            end
            if (outgoing == 1)
                link_to_add = outgoing_links(1);
            elseif (outgoing > 1) % choose the first, cheapest
                [~,indx] = min(curr_orig_link_cost(outgoing_links));
                link_to_add = outgoing_links(indx);
            else % implies no outgoing, incoming only
                [~,indx] = min(curr_orig_link_cost(incoming_links));
                link_to_add = incoming_links(indx);
            end
        else % Phase 2 link selection differs from Phase 1
            % Original method Cheapest INCOMING First (Phase 2)
            outgoing = 0; outgoing_links = [];
            incoming = 0; incoming_links = [];
            for x = 1:1:count_extra_basic_link % sort links
                link = extra_basic_links(x);
                inode = node_i(link);
                jnode = node_j(link);
                if (inode == node_unknown)% as an outgoing link
                    outgoing = outgoing + 1;
                    outgoing_links(outgoing) = link;
                else % or an incoming link
                    incoming = incoming + 1;
                    incoming_links(incoming) = link;
                end
            end
            if (outgoing == 1)
                link_to_add = outgoing_links(1);
            elseif (outgoing > 1) % choose the first, cheapest
                [~,indx] = min(curr_orig_link_cost(outgoing_links));
                link_to_add = outgoing_links(indx);
            else % implies no outgoing, incoming only
                [~,indx] = min(curr_orig_link_cost(incoming_links));
                link_to_add = incoming_links(indx);
            end
        end
    else % Phase 2 link selection differs from Phase 1
        % Original method Cheapest INCOMING First (Phase 2)
        outgoing = 0; outgoing_links = [];
        incoming = 0; incoming_links = [];
        for x = 1:1:count_extra_basic_link % sort links
            link = extra_basic_links(x);
            inode = node_i(link);
            jnode = node_j(link);
            if (inode == node_unknown)% as an outgoing link
                outgoing = outgoing + 1;
                outgoing_links(outgoing) = link;
            else % or an incoming link
                incoming = incoming + 1;
                incoming_links(incoming) = link;
            end
        end
        if (outgoing == 1)
            link_to_add = outgoing_links(1);
        elseif (outgoing > 1) % choose the first, cheapest
            [~,indx] = min(curr_orig_link_cost(outgoing_links));
            link_to_add = outgoing_links(indx);
        else % implies no outgoing, incoming only
            [~,indx] = min(curr_orig_link_cost(incoming_links));
            link_to_add = incoming_links(indx);
        end
    end
incoming_links(incoming) = link;
end
if (incoming == 1)
  link_to_add = incoming_links(1);
elseif (incoming > 1) % choose the first, cheapest
    [~,indx] = min(curr_orig_link_cost(incoming_links));
    link_to_add = incoming_links(indx);
else % implies no incoming, only outgoing links
    [~,indx] = min(curr_orig_link_cost(outgoing_links));
    link_to_add = outgoing_links(indx);
end
else % only have one extra link to pull in
    link_to_add = extra_basic_links(count_extra_basic_link);
end
fprintf(outFileID,'Bringing in %d-%d
', node_i(link_to_add), node_j(link_to_add));
% move non-basic to basic set and update node arrays
basic_links = [basic_links (basic_links(end) + 1)];
num_basic_links = num_basic_links + 1;
onbasic_links(end) = [];
num_nonbasic_links = num_nonbasic_links - 1;
node_i = [node_i (node_i(link_to_add))];
node_i(link_to_add) = [];
node_j = [node_j (node_j(link_to_add))];
node_j(link_to_add) = [];
link_cost = [link_cost (link_cost(link_to_add))];
link_cost(link_to_add) = [];
if(phase == 1)
    link_values = [link_values curr_orig_link_cost(link_to_add)];
    link_values(link_to_add) = [];
end
curr_orig_link_cost(link_to_add) = [];
link_flows = [link_flows (link_flows(link_to_add))];
link_flows(link_to_add) = [];
end
end
num_iterations = 2; % this is a minimum for quick check of weights
end % End of MCFStep1 function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% F_Sub1MCFStep1                                                    %
% Overview: This sub_function will calculate the node weight values   %
% Calling functions: MCFStep1                                         %
% Inputs:  link_cost - an array of costs associated with each link     %
%         node_i - an array of node values of i-end links             %
%         node_j - an array of node values of j-end links             %
% num_basic_links - the number of basic links
% num_iterations - maximum number of iterations to perform
% num_nonbasic_links - the number of non-basic links
% In/Out: w - an array of weights for each node

function [w] = F_Sub1MCFStep1(link_cost, node_i, node_j, num_basic_links, num_iterations, num_nonbasic_links, w)
% calculate as many weights as possible
for iteration = 1:num_iterations
    for link = (num_nonbasic_links + 1):\(\text{num_nonbasic_links + num_basic_links})\:
        if (w(node_i(link)) == -9999 & w(node_j(link)) ~ -9999)
            w(node_i(link)) = w(node_j(link)) + link_cost(link);
        elseif(w(node_j(link)) == -9999 & w(node_i(link)) ~ -9999)
            w(node_j(link)) = w(node_i(link)) - link_cost(link);
        end
    end
    if (size(find(w == -9999),2) == 0) % no additional missing weights
        break;
    end
end % End of Sub1MCFStep1 function
A.2. Conventional MCF Method, Step #2

\[function\]

\[
\begin{align*}
\text{all_negative} &= \text{false}; \% \text{ for Phase 2} \\
\text{max_pos_value} &= 0; \% \text{ initialize the max positive value to zero} \\
\text{fprintf(outFileID, 'Step2 [ ')}; \\
\text{for } i = 1:\text{num_nonbasic_links} \% \text{ find the maximum positive value} \\
\text{link_nonbasic} &= \text{nonbasic_links}(i); \\
\text{value}_1 &= w(\text{node}_i(link_nonbasic)); \\
\text{value}_2 &= w(\text{node}_j(link_nonbasic)); \\
\text{value} &= \text{value}_1 - \text{value}_2 - \text{link_cost}(\text{link_nonbasic}); \\
\text{fprintf(outFileID, ' %3d', value); \\
\text{if (value > 0) && value > max_pos_value} \\
\text{max_pos_value} &= \text{value}; \\
\text{end} \\
\end{align*}
\]

% display status information
\text{fprintf(outFileID,'\n');}
\text{F_PrintStatus(node_i,node_j,link_flows,outFileID);} \\
\text{fprintf(outFileID,'Current Objective Value = %d', max_pos_value);
if (max_pos_value == 0) % no positive values were found
  link_enter = 0;  %% All links are negative
  all_negative = true; % for Phase 2 - indicates STOP
  return;
else % at least one candidate non-basic link to consider for basic set
  icount = 0; % initialize icount variable to zero
  for i = 1:1:num_nonbasic_links % find all link candidates
    link_nonbasic = nonbasic_links(i);
    valuei = w(node_i(link_nonbasic));
    valuej = w(node_j(link_nonbasic));
    % use the formula 'v = wi - wj - cij' in Step2
    value = valuei - valuej - link_cost(link_nonbasic);
    if (value == max_pos_value)
      icount = icount + 1;
      link_enter_candidates(icount) = link_nonbasic;
    end
  end
  % if there is a tie, selection is based on minimum cost
  link_min_cost = 9876543210;  % arbitrarily large number
  for i = 1:1:icount % if there is a second tie, take first link
    cost_link = curr_orig_link_cost(link_enter_candidates(i));
    if (cost_link < link_min_cost)
      link_min_cost = cost_link;
      link_select = link_enter_candidates(i);
    end
  end
  link_enter = link_select; % link to enter the basic set
  % move non-basic to basic set, update node arrays & curr_orig cost
  basic_links = [basic_links (basic_links(end) + 1)];
  num_basic_links = num_basic_links + 1;
  nonBasic_links(end) = [];
  num_nonbasic_links = num_nonbasic_links - 1;
  node_i = [node_i (node_i(link_enter))];
  node_i(link_enter) = [];
  node_j = [node_j (node_j(link_enter))];
  node_j(link_enter) = [];
  if (phase == 1)
    link_cost(end + 1) = 0;         % this is only for phase 1
  else
    link_cost = [link_cost (link_cost(link_enter))]; % phase 2
  end
  link_cost(link_enter) = [];
  link_values = [link_values curr_orig_link_cost(link_enter)];
  link_values(link_enter) = [];
  curr_orig_link_cost(link_enter) = [];
  link_flows = [link_flows link_flows(link_enter)];
  link_flows(link_enter) = [];
end
% display status information
fprintf(outFileID,' Function (MCFStep2) has completed\n');
fprintf(outFileID,' Link to enter basic group is %d-%d\n', ...
  node_i(end), node_j(end));
end % End of MCFStep2 function
A.3. Conventional MCF Method, Step 3

% Minimum Cost Flow (MCF) Problem
% Jointly written by Timothy M. Chavez and Duc Thai Nguyen
% Email: ChavezTM@Gmail.com; iPhone # 757-561-0778
% Email: DNguyen@ODU.edu; iPhone # 757-560-0113
% Original/Start Date: Spring Semester 2021 -version: 210612:1300
% F_MCFStep3
% Overview: This function will use the loop identified in F_IDLinksInLoop to identify the basic link that should be removed from the basic set by first calculating the theta value
% Calling functions: ProcessMethod2 and ProcessMethod3
% Inputs: num_nonbasic_links - the number of non-basic links
% outFileID - file handle to direct output (1=screen)
% Outputs: theta - calculated value of theta used to remove basic link
% In/Out: basic_links - an array to identify the basic links
% link_flows - an array of flows along the links
% link_status - the status "in/out more/less" of the loop links
% node_i - an array of node values of i-end links
% node_j - an array of node values of j-end links
% num_basic_links - the number of basic links
% link_cost - an array of costs associated with each link
% curr_orig_link_cost - current (dynamic) original link costs
% orig_link_cost - original link costs (unmodified)
% link_values - phase 1 artificial link costs
% function
% Determine the set of links in the shortest loop
Tnode_i = node_i((num_nonbasic_links + 1):end);
Tnode_j = node_j((num_nonbasic_links + 1):end);
links_loop = F_IDLinksInLoop(Tnode_i, Tnode_j);
if (isempty(links_loop))
    link_status = [];
    theta = [];
    return;
end
% Display status information
fprintf(outFileID,'     links in loop [ ');
for indx = 1:size(links_loop,2)
fprintf(outFileID,'%d-%d ',
    node_i(num_nonbasic_links + links_loop(indx)),
    node_j(num_nonbasic_links + links_loop(indx)));
end
fprintf(outFileID,'
');
% initialize variable before traveling loop to identify link statuses
smallest_theta = 9876543210; % set a very large number
num_links_loop = numel(links_loop);
loose_inode = node_i(end);  % link entered from Step2
loose_jnode = node_j(end);  % link entered from Step2
% status_loose_Xnode = {1=out_more; 2=out_less; 3=in_more; 4=in_less}
status_loose_inode = 1;  % initial link is i to j with flow = theta
status_loose_jnode = 3;  % link is incoming to j with a flow increase
% examine each link in the loop to determine their link status
for iteration = 1:1:num_links_loop % arbitrary # of iterations
    fprintf(outFileID,'   Updating Loop Node Status: Cycle - %d\n', ...
        iteration);
    for i = 1:1:(num_links_loop - 1) % # of other links in closed loop
        fprintf(outFileID,'     link #%d in loop\n',i);
        link_num = links_loop(i);
        inode = node_i(link_num + num_nonbasic_links);
        jnode = node_j(link_num + num_nonbasic_links);
        if (inode ~= loose_inode  &&  inode ~= loose_jnode  && ...
            jnode ~= loose_inode  &&  jnode ~= loose_jnode)
            fprintf(outFileID,'     NOT connected to loose node\n');
            link_status(i) = 5;  % mark link not complete
            continue;  % because node being examined is not an end node
        end
        % There are 4 cases to consider when determining theta
        % case 1 of 4: link_loop is connected to loose_inode as incoming link
        % we are considering in terms of the current loose_inode
        if (jnode == loose_inode)
            fprintf(outFileID,'     need to update status_loose_inode\n');
            fprintf(outFileID,' & link_status\n');
            if (status_loose_inode == 1 || status_loose_inode == 4)
                status_loose_inode = 1;  % this will be new loose_inode
                link_status(i) = 3;  % analyzed at current loose_inode
                % in this case the status is "more" or +Theta
                % Theta can be any value and we can ignore it
                fprintf(outFileID,'     flow(link_num) + theta .GE. \n\n\n\n');
                fprintf(outFileID,'     WARNING: REVERSE link DIRECTION!\n');
            elseif (theta > 0  && theta < smallest_theta)
                smallest_theta = theta;
            end
        else % when the current_loose_inode is a 2 or a 3
            status_loose_inode = 2;  % for the new loose_inode
            link_status(i) = 4;  % in terms of current_loose_inode
            % in this case the status is "less" or -Theta and we
            % have to see if it is now the smallest positive value
            theta = link_flows(link_num + num_nonbasic_links);
            if (theta == 0)
                fprintf(outFileID,'     WARNING: \n');
                fprintf(outFileID,'     REVERSE link DIRECTION!\n');
            else % if we have to check to see if this is a link % to/from the artificial link or not... could be % an issue if it is not the artificial link
                node_i(link_num + num_nonbasic_links) = jnode;
                node_j(link_num + num_nonbasic_links) = inode;
                status_loose_jnode = 3;  % reversed
                link_status(i) = 1;  % reversed
            end
        end
        % now we update the the new_loose_inode
        fprintf(outFileID,'   need to update \n');
        fprintf(outFileID,'   loose_inode = inode\n');
        loose_inode = inode;  % new end node
% There are 4 cases to consider when determining theta
% case 2 of 4: link_loop is connected to loose_inode as out_going link
% we are considering in terms of the current loose_inode
elseif (inode == loose_inode)
    fprintf(outFileID,' need to update ');
    fprintf(outFileID,'status_loose_inode ');
    fprintf(outFileID,'% link_status
');
    if (status_loose_inode == 2 || status_loose_inode == 3)
        status_loose_inode = 3; % this will be new loose_inode
        link_status(i) = 1; % analyzed at current loose_inode
        if in this case the status is "more" or +Theta
            Theta can be any value and we can ignore it
        fprintf(outFileID,' flow(link_num) + theta .GE. ');
        fprintf(outFileID,'zero --> ignore
');
    else % when the current_loose_inode is a 1 or a 4
        status_loose_inode = 4; % for the new loose_inode
        link_status(i) = 2; % in terms of current loose_inode
        if in this case the status is "less" or -Theta and we
            have to see if it is now the smallest positive value
        theta = link_flows(link_num + num_nonbasic_links);
        if (theta == 0)
            fprintf(outFileID,' WARNING: ');
            fprintf(outFileID,'REVERSE link DIRECTION!
');
            if we will have to check to see if this is a link
                to/from the artificial link or not... could be
                an issue if it is not the artificial link
        node_i(link_num + num_nonbasic_links) = jnode;
        node_j(link_num + num_nonbasic_links) = inode;
        status_loose_jnode = 1; % reversed
        link_status(i) = 3; % reversed
        elseif (theta > 0  && theta < smallest_theta)
            smallest_theta = theta;
    end
else % when the current_loose_inode is a 2 or a 3
    status_loose_inode = 2; % for the new loose_inode
    link_status(i) = 4; % in terms of current loose_inode
    if in this case the status is "less" or -Theta and we
        have to see if it is now the smallest positive value
% now we update the the new loose_inode
% There are 4 cases to consider when determining theta
% case 3 of 4: link_loop is connected to loose_jnode as in_coming link
% we are considering in terms of the current loose_jnode
elseif (jnode == loose_jnode)
    fprintf(outFileID,' need to update ');
    fprintf(outFileID,'status_loose_jnode ');
    fprintf(outFileID,'% link_status
');
    if (status_loose_jnode == 1 || status_loose_jnode == 4)
        status_loose_jnode = 1; % this will be new loose_inode
        link_status(i) = 3; % analyzed at current loose_inode
        if in this case the status is "more" or +Theta
            Theta can be any value and we can ignore it
        fprintf(outFileID,' flow(link_num) + theta .GE. ');
        fprintf(outFileID,'zero --> ignore
');
    else % when the current_loose_inode is a 2 or a 3
        status_loose_jnode = 2; % for the new loose_jnode
        link_status(i) = 4; % in terms of current loose_inode
        if in this case the status is "less" or -Theta and we
            have to see if it is now the smallest positive value

theta = link_flows(link_num + num_nonbasic_links);
if (theta == 0)
    fprintf(outFileID,' WARNING: ');
    fprintf(outFileID,'REVERSE link DIRECTION!
');
    % we will have to check to see if this is a link
    % to/from the artificial link or not... could be
    % an issue if it is not the artificial link
    node_i(link_num + num_nonbasic_links) = jnode;
    node_j(link_num + num_nonbasic_links) = inode;
    status_loose_jnode = 3; % reversed
    link_status(i) = 1; % reversed
    elseif (theta > 0  && theta < smallest_theta)
        smallest_theta = theta;
    end

% now we update the the new loose_inode
fprintf(outFileID,' need to update loose_jnode = inode
');
loose_jnode = inode; % new end node

% There are 4 cases to consider when determining theta
% case 4 of 4: link_loop is connected to loose_jnode as out_going link
% we are considering in terms of the current loose_inode
elseif (inode == loose_jnode)
    fprintf(outFileID,' need to update ');
    fprintf(outFileID,'status_loose_jnode ');
    fprintf(outFileID,'& link_status
');
    if (status_loose_jnode == 2 || status_loose_jnode == 3)
        status_loose_jnode = 3; % this will be new loose_inode
        link_status(i) = 1; % analyzed at current loose_inode
        % in this case the status is "more" or +Theta
        % Theta can be any value and we can ignore it
        fprintf(outFileID,' flow(link_num) + theta .GE. ');
        fprintf(outFileID,'zero --> ignore
');
    else % when the current_loose_inode is a 2 or a 3
        status_loose_jnode = 4; % for the new loose_inode
        link_status(i) = 2; % in terms of current loose_inode
        % in this case the status is "less" or -Theta and we
        % have to see if it is now the smallest positive value
        theta = link_flows(link_num + num_nonbasic_links);
        if (theta == 0)
            fprintf(outFileID,' WARNING: ');
            fprintf(outFileID,'REVERSE link DIRECTION!
');
            % we will have to check to see if this is a link
            % to/from the artificial link or not... could be
            % an issue if it is not the artificial link
            node_i(link_num + num_nonbasic_links) = jnode;
            node_j(link_num + num_nonbasic_links) = inode;
            status_loose_jnode = 1; % reversed
            link_status(i) = 3; % reversed
        elseif (theta > 0  && theta < smallest_theta)
            smallest_theta = theta;
        end
    end

% now we update the the new loose_inode
fprintf(outFileID,' need to update ');
fprintf(outFileID,'loose_jnode = jnode
');
loose_jnode = jnode; % new end node
end
% check to make sure the basic links LOOP is completely closed
% otherwise, go back to iteration loop !!
if (loose_inode == loose_jnode)
    link_status(end+1) = 1;
    break; % the loop is closed so done with this iteration
end
end
if(sum(find(link_status == 5)) == 0)
    break; % there are no uncalculated links
end
end
% display final value calculated for theta
theta = smallest_theta;
% Display status information
fmt=['   Status =' repmat(' %d ',1,numel(link_status)) '
'];
fprintf(outFileID,fmt,link_status);
% Update link flow based on the determined theta value
fprintf(outFileID,'       Flow updates [ ');
for i = 1:1:num_links_loop
    link_num = links_loop(i);
    if(link_status(i) == 1 || link_status(i) == 3) % "in/out more"
        link_flows(link_num + num_nonbasic_links) = ...
        link_flows(link_num + num_nonbasic_links) + theta;
    else % if link_status is a 2 or a 4 then "in/out less"
        link_flows(link_num + num_nonbasic_links) = ...
        link_flows(link_num + num_nonbasic_links) - theta;
    end
    fprintf(outFileID,'%d-%d=%d ',node_i(link_num + num_nonbasic_links), node_j(link_num + num_nonbasic_links), link_flows(link_num + num_nonbasic_links));
end
fprintf(outFileID,'
');
% Now we need to UPDATE basic_links and Reset basic links to positive
removed = 0;  %could run for loop backwards and not need "removed"
links_loop_ascending = sort(links_loop,'ascend');
for i = 1:1:num_links_loop
    link_num = links_loop_ascending(i);
    if (link_flows(link_num - removed + num_nonbasic_links) == 0)
        fprintf(outFileID,' Remove link: %d-%d\n',
        node_i(link_num - removed + num_nonbasic_links), node_j(link_num - removed + num_nonbasic_links));
        node_i(link_num - removed + num_nonbasic_links) = [];
        node_j(link_num - removed + num_nonbasic_links) = [];
        basic_links(end) = [];
        num_basic_links = num_basic_links - 1;
        link_values(link_num - removed + num_nonbasic_links) = [];
        link_flows(link_num - removed + num_nonbasic_links) = [];
        link_cost(link_num - removed + num_nonbasic_links) = [];
        removed = removed + 1;
    end
end
% Display status information
fprintf(outFileID,' Function (MCFStep3) has completed\n');
fprintf(outFileID,'     smallest, positive value %d
',theta);
end % End of MCFStep3 function


ATTACHMENT B – MCF SOLUTIONS BY-HAND

The following MS PowerPoint presentations detail the by-hand solutions for some of the conventional MCF examples called out in this work.
B.1. Conventional MCF By-Hand Solution for Chapter V, Example #1

Link to MS PowerPoint Presentation → https://tinyurl.com/2pms48cw

Example #1 [9-Node/14 Link Network] — MCF Phase 1 / MCF Phase 2

Objective is to find the number of items on each link such that the cost is minimized and equilibrium is maintained

\[
\min \{b(l)\} \quad \text{subject to} \\
\sum_{j} x(jl) = x(lj)
\]

Equilibrium Constraints:

- Node 1:
  \[ x(21) + x(31) - x(13) = 3 \]

- Node 2:
  \[ x(12) + x(32) - x(23) = 0 \]

- Node 3:
  \[ x(23) - x(35) = 3 \]

- Node 4:
  \[ x(34) - x(45) = 0 \]

- Node 5:
  \[ x(45) + x(51) - x(59) = 0 \]

- Node 6:
  \[ -x(59) + x(56) = 0 \]

- Node 7:
  \[ x(67) + x(76) = 3 \]

- Node 8:
  \[ x(68) + x(86) = 0 \]

- Node 9:
  \[ x(98) - x(89) = 5 \]

Phase I - Find the initial, feasible solution from the network above.

Current Flow:

- \[ x_{12} = 0 \text{ (0)} \]
- \[ x_{21} = 0 \text{ (0)} \]
- \[ x_{23} = 0 \text{ (0)} \]
- \[ x_{32} = 0 \text{ (0)} \]
- \[ x_{43} = 0 \text{ (0)} \]
- \[ x_{54} = 0 \text{ (0)} \]
- \[ x_{59} = 0 \text{ (0)} \]
- \[ x_{67} = 0 \text{ (0)} \]
- \[ x_{68} = 0 \text{ (0)} \]
- \[ x_{76} = 0 \text{ (0)} \]
- \[ x_{86} = 0 \text{ (0)} \]
- \[ x_{98} = 0 \text{ (0)} \]

Phase II - Draw the new network with an extra "bump" node (node A) and the links to the original (9) nodes. Assign costs to the (solid lines) new basic links = 1 and set the (dotted lines) original links = 0 (they become non-basic links).

Initial cost = \[ C_{1} = x_{12} + x_{23} + x_{32} + x_{43} + x_{54} + x_{59} + x_{67} + x_{68} + x_{76} + x_{86} + x_{98} = 0 \]

Step 2:
Determine which non-basic variable should be added to the basic group.

- \[ x_{13} = x_{14} = x_{23} = x_{24} = x_{34} = x_{45} = x_{56} = x_{57} = x_{68} = x_{78} = x_{89} = x_{98} = 0 \]

Need to break tie for \[ x_{13}, x_{75}, \text{ and } x_{78} \]

using original cost value. Pick least cost.

- \[ C_{13} = 6, C_{75} = 2, C_{78} = 3 \]

Pick \[ x_{78} \] to add to basic group.

Iteration 1 - Step 1:
Find \( w_{l} = \min \{b_{l}\} \) (arbitrarily) using basic links and \[ w_{l} = x_{l} - c_{l} \]

- \[ w_{1} = x_{1} = -1 \]
- \[ w_{2} = x_{2} = -1 \]
- \[ w_{3} = x_{3} = -1 \]
- \[ w_{4} = x_{4} = -1 \]
- \[ w_{5} = x_{5} = -1 \]
- \[ w_{6} = x_{6} = -1 \]
- \[ w_{7} = x_{7} = -1 \]
- \[ w_{8} = x_{8} = -1 \]
- \[ w_{9} = x_{9} = -1 \]

\[ w_{l} = x_{l} - c_{l} \]
Step 1:
Determine which basic variable to remove. Consider X75 = @ then X74 = @ - 1@ and X60 = @ - 1@. Such that:
X75 is @ = 0
X74 is @ = 0 or @ = 0
X60 is @ = 1 or @ = -1
Choosing map = 3 will satisfy all of the above and result in X75 = 3, X74 = 0, and X60 = -3 so X75 will be dropped out of the basic group and X60 will be reversed.
Recalculate cost:
\[C \text{cost} = \text{C17A09} + \text{C36YFA} + \text{C35A06} + \text{Y35Y75} + \text{C65YAB} + \text{C34A06} + \text{B34B15} = \text{cost of nodes} + 15 + 15 + 15 = 0 + 15 + 15 = 0 + 15 + 15 = 15\]

Determine flow values for loop:
Given X75 as @ then flow at node 7 is "out more" and flow at node 5 is "in more"
Looking at flow along X73 must be "more" or "less".
Looking at flow along X74 must be "less" or "more".
Looking at flow along X75 must be "less" or "more" so flow at node 5 is "out less" so flow at node 5 is "out less" to flow along X54 must be "less" or "more".

Current flow:
X51 = 60 = +30
X52 = 0 = +30
X53 = 0 = 0
X54 = 0 = 0
X55 = 0 = 0
X56 = 0 = 0
X57 = 0 = 0
X58 = 0 = 0
X59 = 60 = +30

Iteration II - Step 1:
Find (Ax - Aw) with Ax = @ (arbitrarily) using basic links and \[\text{Ax} = @ - \text{Ax} - \text{Aw}\]
\[\text{Aw} = @ + @ + @ = -1\]
\[\text{Ax} = @ + @ + @ = 0\]
\[\text{Aw} = @ + @ + @ = 0\]
\[\text{Aw} = @ + @ + @ = 0\]
\[\text{Aw} = @ + @ + @ = 0\]
\[\text{Aw} = @ + @ + @ = 0\]

Step 2:
Determine which non-basic variable should be added to the basic group:
X12: \(v1 = v2 = c13 = 0\)
X14: \(v1 = v2 = c13 = 0\)
X23: \(v2 = v3 = c23 = 0\)
X35: \(v3 = v4 = c35 = 0\)
X47: \(v4 = v7 = c47 = 0\)
X51: \(v5 = v11 = c51 = 1\)
X53: \(v6 = v7 = c53 = 1\)
X61: \(v6 = v9 = c61 = 0\)
X78: \(v7 = v8 = c78 = 1\)
X85: \(v8 = v9 = c85 = 0\)
X89: \(v9 = v8 = c89 = 0\)
X99: \(v9 = v8 = c99 = 0\)

Need to break tie for X51, X59, and X78 using original cost value. Plot least cost.
\[\text{c13} = 6\]
\[\text{c15} = 4\]
\[\text{c78} = 9\]
Pick X78 to add to basic group.
Step 1: Determine which basic variable to remove. Consider $X7B = 0$ then $X7S = 3 \geq 0$, $X5A = 3 \geq 0$, and $XAB = 0 \geq 0$.

Such that:
- $X7B = 0$
- $X7S = 3 \geq 0 \Rightarrow \text{or} = 0$
- $X5A = 0 \leq 0 \Rightarrow \text{or} = 0$
- $XAB = 0 \leq 0 \Rightarrow \text{or} = 0$

Choosing $X7B = 0$ will satisfy all of the above and result in $X7B = 3$, $X7S = 0$, $X5A = 0$, and $XAB = 0$ so both $X7S$ and $X5A$ will drop out of the basic group and $XAB$ will be reversed.

Reinitialize cost:
- $C(X7B) + C(X7S) + C(X5A) + C(XAB) = C(X7B) + C(X5A) + 3 * 0 + 0 * 0$
- $1 + 1 + 6 + 0 = 8$
- $x = 1$

Determining flow values for loop:

Given $X7B$ as $0$ then flow at node $7$ is "out more" and flow at node $8$ is "in more" so flow along $X7B$ must be "more" or "less".

Looking at $X5A$ we see flow at node $5$ is "out less" and flow at node $6$ is "in less" so flow along $X5A$ must be "less" or "more".

Looking at $XAB$ we see flow at node $A$ is "out less" so flow at node $A$ is "out less" so flow along $XAB$ must be "less" or "more".

Current flow:
- $XAB = 0 = +0$
- $X5A = 0 = -0$
- $X7B = 0 = +0$
- $X7S = 0 = +0$
- $X5A = 0 = +0$

Step 2: Determine which non-basic variable should be added to the basic group.

Need to break tie for $X35$, $X6$, and $X60$ using original cost value.

Pick $X35$ to add to basic group.

Iteration III - Step 1: Find $(xB - uB)$ with $xB = 0$ (arbitrarily) using basic links and $uB = -0.01$

$uB = w1 = 1$, $w1 = -1$
$w1 = w2 = 1$, $w2 = -1$
$w1 = w3 = 1$, $w3 = -1$
$w1 = w4 = 1$, $w4 = -1$
$w1 = w5 = 1$, $w5 = -1$
$w1 = w6 = 1$, $w6 = -1$
$w1 = w7 = -1$, $w7 = 1$
$w1 = w8 = -1$, $w8 = 1$
$w1 = w9 = -1$, $w9 = 1$

Need to bring in link to solve for $w6$ (cheapest outgoing)

$uB = w6 = 0$, $w6 = -1$, $w6 = -1$
Step 1:
Determine which basic variable to remove. Consider XBX = 0 then XSEG = E + G, XBA = 3 - 0, and XAB = 3 - 0.
Such that:
XAB = 3 - 0
XSEG = 0 + G
XBA = 3 - 0 or G
XAB = 3 - G
Choosing odd = 3 will satisfy all of the above and result in XBS = 3, XSB = 3, XBA = 0, and XAB = 3 so XBA will drop out of the basic group.

Recalculate cost:
- CSEG = 
  - CS5 = CS5
  - 0 = CS5
  - CS5 = CS5
  - 78 + CS5
  = 78
  - CS5
  - 78
  = 0
  = 10

Determining flow values for loop:
Given XBS = 3 then flow at node 0 is "out more" and flow at node 5 is "in more".
Looking at flow XSEG = 78, flow at node 3 is "in more" and flow at node 5 is "out more",
so flow along XSB must be "more" or "less".
Looking at flow XBA = 0, flow at node 0 is "out less" and flow at node 6 is "in less",
so flow along XBA must be "less" or "-gz".
Looking at flow XAB = 0, flow at node 0 is "out less" as flow at node A is "out less",
so flow along XAB must be "less" or "-gz".

Current flow:
XAB = 3 - 0
XBA = 0 + 3
XAB = 0 + 3
XBA = 0 + 3
XAB = 0 + 3
XAB = 0 + 3
XAB = 0 + 3
XAB = 0 + 3
XAB = 0 + 3
XAB = 0 + 3
XAB = 0 + 3

Step 2:
Determine which non-basic variable should be added to the basic group.

Pick XBS to add to basic group.

Iteration IV - Step 1:
Find (u, a) such that a = 0 (arbitrarily) using basic links and wi (a, j) cij:

Pick XBS to add to basic group.
Step 3:
Determine which basic variable to remove. Consider \( X15 \geq 0 \) then \( X19 = 3 \geq 0, X16 = 2 \geq 0 \), and 
\( X34 \geq 5 \geq 0 \).

Such that:
\( X15 \leq 0 \)
\( X19 = 3 \geq 0 \)
\( X16 = 2 \geq 0 \)
\( X34 = 5 \geq 0 \)

Choosing \( \alpha = 2 \) will satisfy all of the above and result in \( X15 = 2, X19 = 5, X16 = 0 \), and \( X34 = 3 \) so \( X19, X34 \) dropped out of the basic group.

Recalculate cost
- \( C19B19 = C34B34 \)
- \( C29B29 = C54B54 \)
- \( C39B39 = C64B64 \)
- \( C49B49 = C84B84 \)
- \( C59B59 = C94B94 \)
- \( B19 + B34 = 2 \)
- \( B29 + B54 = 0 \)
- \( B39 + B64 = 0 \)
- \( B49 + B84 = 0 \)
- \( \alpha = 0 \)

Determining flow values for loop:
- Given \( X15 \geq 0 \) then flow at node 9 is “in more” and flow at node 5 is “in more”

Looking at flow along \( X19 \) must be “more” or “less”

Looking at flow along \( X34 \) must be “more” or “less”

Looking at flow along \( X15 \) must be “less” or “less”

Looking at flow along \( X16 \) must be “less” or “less”

Current flow
\( X16 = 0 \)
\( X19 = 0 \)
\( X34 = 0 \)
\( X54 = 0 \)

Step 2:
Determine which non-basic variable should be added to the basic group
\( X11: v_1 = w_2 - C12 \)
\( X13: v_1 = w_1 - C13 \)
\( X15: v_1 = w_1 - C15 \)
\( X16: v_1 = w_2 - C16 \)
\( X17: v_2 = w_1 \)
\( X18: v_2 = w_1 - C18 \)
\( X19: v_2 = w_1 - C19 \)

\( C12 = 6 \)
\( C13 = 1 \)
\( C15 = 1 \)

Need to break tie for \( X35 \) and \( X16 \) using original cost value, Pick least cost.

USI = 6
USI = 1

Pick \( X16 \) to add to basic group
Step 1:
Determine which basic variable to remove. Consider X66 = 0 then X66 = - 1, X68 = S + 1, X76 = 1 + X, and X86 = 2 + X.

Such that:
X66 is 0 or 1
X68 is S + 1 or S + 2 or S + 3 or S + 4
X76 is 1 or X
X86 is 2 or 3

Choosing map 4 will satisfy all of the above and result in X66 = 1, X68 = 1, X76 = X, and X86 = 0 so XAA will be dropped out of the basic group and XAA will be reversed.

Recalculate cost:
\[
\text{Cost} = C_{10} + C_{62} + C_{64} + C_{68} + C_{70} + C_{72} + C_{74} + C_{76} + C_{78} + C_{80} + C_{82} + C_{84} + C_{86} + C_{90} + \text{Cost}\text{Rest of nodes}
\]
\[
= 0 + 2 + 0 + 0 + 1 + 0 + 2 + 0 + 2 + 0 + 2 + 0 + 1 + 0 + 0 = 0
\]

Determining flow values for loop:
Given XAA as 0 then Flow at node 9 is "out more" and Flow at node 1 is "in more".

Looking at Flow Along XAA must be "more" or "less".

Looking at Flow at node 9 is "out more" and Flow at node 1 is "in more".

Looking at XAA must be "more" or "less".

Looking at XAA must be "more" or "less".

Looking at Flow Along XAA must be "more" or "less".

Looking at Flow Along XAA must be "more" or "less".

Looking at Flow Along XAA must be "more" or "less".

Looking at Flow Along XAA must be "more" or "less".

Looking at Flow Along XAA must be "more" or "less".

Current flow:

X16 = 00 + 0
X24 = 00 + 0
X36 = 00 + 0
X46 = 00 + 0
X56 = 00 + 0
X68 = 00 + 0
X76 = 00 + 0
X86 = 00 + 0
X96 = 00 + 0

Iteration VI - Step 1:
Find (u, w) with u = 0 (arbitrarily) using basic links and u(i,v) to (i,v).

\[
\begin{align*}
\text{u1} &= 0 + 1, \text{v1} = 0 \\
\text{u2} &= 1 + 2, \text{v2} = 0 \\
\text{u3} &= 0 + 1, \text{v3} = 0 \\
\text{u4} &= 1 + 2, \text{v4} = 0 \\
\text{u5} &= 0 + 1, \text{v5} = 0 \\
\text{u6} &= 1 + 2, \text{v6} = 0 \\
\text{u7} &= 0 + 1, \text{v7} = 0 \\
\text{u8} &= 1 + 2, \text{v8} = 0 \\
\text{u9} &= 0 + 1, \text{v9} = 0 \\
\end{align*}
\]

Pick X51 to add to basic group

X31 - u1 = C12
X51 - u1 = C23
X61 - u1 = C34
X71 - u1 = C41
X81 - u1 = C51
X91 - u1 = C61
X12 - u2 = C13
X21 - u2 = C23
X32 - u2 = C34
X42 - u2 = C41
X52 - u2 = C51
X62 - u2 = C61
X72 - u2 = C71
X82 - u2 = C81
X92 - u2 = C91
X13 - u3 = C12
X23 - u3 = C23
X33 - u3 = C34
X43 - u3 = C41
X53 - u3 = C51
X63 - u3 = C61
X73 - u3 = C71
X83 - u3 = C81
X93 - u3 = C91

Step 2:
Determine which non-basic variable should be added to the basic group.

Pick X51 to add to basic group

\[
\begin{align*}
\text{X31} &= u1 - w1 = C12 \\
\text{X51} &= u1 - w1 = C23 \\
\text{X61} &= u1 - w1 = C34 \\
\text{X71} &= u1 - w1 = C41 \\
\text{X81} &= u1 - w1 = C51 \\
\text{X91} &= u1 - w1 = C61 \\
\end{align*}
\]

\[
\text{X32} &= u2 - w2 = C13 \\
\text{X42} &= u2 - w2 = C23 \\
\text{X52} &= u2 - w2 = C34 \\
\text{X62} &= u2 - w2 = C41 \\
\text{X72} &= u2 - w2 = C51 \\
\text{X82} &= u2 - w2 = C61 \\
\text{X92} &= u2 - w2 = C71 \\
\text{X13} &= u3 - w3 = C12 \\
\text{X23} &= u3 - w3 = C23 \\
\text{X33} &= u3 - w3 = C34 \\
\text{X43} &= u3 - w3 = C41 \\
\text{X53} &= u3 - w3 = C51 \\
\text{X63} &= u3 - w3 = C61 \\
\text{X73} &= u3 - w3 = C71 \\
\end{align*}
\]
Step 3: Determine which basic variable to remove. Consider X31 = $\delta$ then X31 = $\delta$, X39 = $\delta$, and X6A = $\delta$.

Such that:

X31 is $\delta$ or $\delta$

X39 is $\delta$ or $\delta$

X6A is $\delta$ or $\delta$

Choosing $\delta$ will satisfy all of the above and result in X31 = X39 = X6A = $\delta$

Recalculate cost:

\[ \text{Cost} = 8 \]

Determining flow values for loop:

Every Xii as $\delta$ then flow at node 2 is "out more" and flow at node 1 is "in more" so flow along X31 must be "more" or "<".

Looking at Xii we see flow at node 1 is "in less" and flow at node 3 is "out less" so flow along X39 must be "less" or "<".

Looking at Xii we see flow at node 1 is "in less" and flow at node 3 is "out less" so flow along X6A must be "less" or "<".

Looking at Xii we see flow at node 2 is "out less" and flow at node 4 is "in less" so flow along X39 must be "less" or "<".

Looking at Xii we see flow at node 2 is "out less" and flow at node 4 is "in less" so flow along X6A must be "less" or "<".

\[ \text{Cost} = 8; \text{STOP PAGE 2} \]

Current flow

<table>
<thead>
<tr>
<th>Node</th>
<th>FlowOut</th>
<th>Supply</th>
<th>Demand</th>
<th>Balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>07</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>09</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\[ \text{Predecessor Value = 0} \]
Phase II - Find the optimal solution from the network produced at the end of Phase I.

Starting cost = Cost of network - Cost of network (Phase I) + Cost of network (Phase II)

Phase I - iteration 1 - Step 1:
First (i.e., x = 0) with x = 0 (first node with most links > 3) using basic links, original costs, and x = 0 if x < 1

x3 = x6 = 0, x3 = 3, x6 = 6
x6 = x9 = 6, x6 = 6
x5 = x5 = 0, x5 = 0
x7 = x8 = 1, x7 = 1

Need to bring in links to solve for x6, x9, and x6 (cheapest incoming)

x2 = x1 = x6 = 1; x6 = 1; x6 = -5

Step 1:
Determine which non-basic variable should be added to the basic group

X31: x2 = x3 = C23 = (-3) = 6 - 1 = -5
X51: x3 = x6 = C26 = (-8) = 6 - 1 = -2
X71: x6 = x8 = C28 = (-5) = 6 - 1 = -2
X75: x6 = x5 = C75 = 4 - 0 = 2

Pick the largest positive value, x8 = 2, to add to basic group

Step 2:
Determine which basic variable to remove. Consider x8 = 2 then X85 <= 0, and x95 <= 0.

Such that:
X85 <= 0
X95 <= 0

Choosing needs 2 will satisfy. All of the above and result in X85 <= 0, X95 <= 0, and X95 = 0 or X55 will dropped out of the basic group.

Recalculate cost

Cost = Cost of network - Cost of network (Phase I) + Cost of network (Phase II)

Phase II = 7

Determining flow values for loop:

Given X85 = 0 then flow at node 8 is "not more" and flow at node 8 is "in more" so flow along X8 must be "move" or "in".

Looking at X85 we see flow at node 8 is "out less" and flow at node 5 is "in less" so flow along X8 must be "less" or "out".

Looking at X95 we see flow at node 9 is "in less" and flow at node 8 is "out less" so flow along X9 must be "less" or "out".
Iteration II - Step 1
First (x1, x2) with x6 = 0 (first node with most links = 3) using basic links, original costs, and x1,x2 < 13

Step 2:
Determine which non-basic variable should be added to the basic group

No positive values exist; stop Phase 2
B.2. Conventional MCF By-Hand Solution for Chapter V, Example #2

Link to MS PowerPoint Presentation ➔ https://tinyurl.com/4jywpcbx

Example #2 (13-Node/20 Link) Network — MCF Phase 1 / MCF Phase 2

Objective is to find the number of items on each link such that the cost is minimized and equilibrium is maintained

Equilibrium Constraints:
- For each node except source and sink:
  \[ \sum_{j \in N(i)} x_{ij} - \sum_{j \in N(i)} x_{ji} = 0 \]

Phase I - Find the initial, feasible solution from the network above.

Current Flow

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 1: Determine which non-basic variable should be added to the basic group

Iteration I - Step 1:

<table>
<thead>
<tr>
<th>Index</th>
<th>w1 - w1</th>
<th>w2 - w2</th>
<th>w3 - w3</th>
<th>w4 - w4</th>
<th>w5 - w5</th>
<th>w6 - w6</th>
<th>w7 - w7</th>
<th>w8 - w8</th>
<th>w9 - w9</th>
<th>w10 - w10</th>
<th>w11 - w11</th>
<th>w12 - w12</th>
<th>w13 - w13</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>-w1 + x1 - w2 = 0</td>
<td>u2</td>
<td>x1 - w2 + x2 = 0</td>
<td>u3</td>
<td>x2 - w3 + x3 = 0</td>
<td>u4</td>
<td>x3 - w4 + x4 = 0</td>
<td>u5</td>
<td>x4 - w5 + x5 = 0</td>
<td>u6</td>
<td>x5 - w6 + x6 = 0</td>
<td>u7</td>
<td>x6 - w7 + x7 = 0</td>
</tr>
</tbody>
</table>

Note: We need to break ties for X112, X123, X139, X1211, X1312, X139, and X139 using original cost value. Pick least cost.
Step 3: Determine which basic variable to remove. Consider $X_{9012} = \theta$ then $X_{1104} \leq 6 \quad \theta$ and $X_{8021} = \lambda$.

Such that:
$X_{9012} = \theta \Rightarrow \theta = 0$
$X_{1104} \leq 6 \quad \theta = 0 \quad \theta \geq 6$
$X_{8021} = \lambda \Rightarrow \theta = 0 \quad \theta \geq -6$

Choosing non-$\theta$ will satisfy all of the above and result in $X_{9012} = 6$, $X_{1104} = 0$, and $X_{8021} = -6$ so $X_{1104}$ will be dropped out of the basic group and $X_{8021}$ will be reversed.

Recalculate cost:
- $C_{1200} + C_{2200} + C_{1101} + C_{1102} + C_{1230} - C_{1100} - C_{1201} = 0$
- $C_{1200} + C_{2200} + C_{1101} + C_{1102} + C_{1230} - C_{1100} - C_{1201} = 0$
- $C_{1200} + C_{2200} + C_{1101} + C_{1102} + C_{1230} - C_{1100} - C_{1201} = 0$
- $C_{1200} + C_{2200} + C_{1101} + C_{1102} + C_{1230} - C_{1100} - C_{1201} = 0$

Determining flow values for loop:
- Given $X_{9012} = \theta$ then flow at node 28 is "not zero" and flow at node 12 is "in zero" so flow along $X_{9012}$ must be "zero" or "+θ".
- Looking at flow along $X_{9012}$ we see flow at node 28 is "not zero" and flow at node 12 is "in zero" so flow along $X_{9012}$ must be "less" or "-θ".

Iteration II - Step 1:
Find (at least) one $w_{ij}$ with $w_{ij} < 0$ (arbitrarily) using basic links and $\text{wd} = \text{nj}$

$w_{ij} = w_{ij} + C_{ij} \times \lambda$; $w_{ij} < 0$
$w_{ij} = w_{ij} + C_{ij} \times \theta$; $w_{ij} \geq 0$
$w_{ij} = w_{ij} + C_{ij} \times \theta$; $w_{ij} \geq 0$

Step 2:
Determine which non-basic variables should be added to the basic group

$X_{1104}$ - $w_{12} - w_{64} - C_{1004} = 1$
$X_{1211} - w_{12} - C_{1211} = 1$
$X_{1201} - w_{12} - C_{1201} = 1$
$X_{1202} - w_{12} - C_{1202} = 1$
$X_{1203} - w_{12} - C_{1203} = 1$
$X_{1212} - w_{12} - C_{1212} = 1$
$X_{1230} - w_{12} - C_{1230} = 1$

Need to break the ties for $X_{1400}$, $X_{1409}$, $X_{1207}$, $X_{1217}$, $X_{1212}$, $X_{1206}$, and $X_{1208}$ using original cost value. Pick least cost:

$X_{1400} = 5$
$X_{1409} = 5$
$X_{1207} = 5$
$X_{1217} = 5$
$X_{1212} = 5$
$X_{1206} = 5$

Need to break ties for $X_{1104}$, $X_{1109}$, and $X_{1208}$ by picking direct occurrences:

Pick $X_{1212}$ to add to basic group
Step 3:
Determine which basic variable to remove. Consider $X_{1211} = \theta$ then $X_{1206} = 6$. $\theta$ and $X_{1201} = 1 - \theta$.
Such that:
$X_{1211} = \theta \Rightarrow \theta = 0$ or $\theta = 1$
$X_{1201} = 1 - \theta \Rightarrow \theta = 0$ or $\theta = 1$
Choosing \( \theta = 0 \) will satisfy all of the above and result in $X_{1211} = 1$, $X_{1206} = 5$, and $X_{1201} = 0$ so $X_{1211}$ will dropped out of the basic group.

Recalculate cost:
\[
\text{Cost} = C_{1204}/X_{1204} + C_{1206}/X_{1206} + C_{1207}/X_{1207} + C_{1211}/X_{1211} + C_{1205}/X_{1205} + \ldots \text{(rest of nodes)}
\]
$\theta = 0$: $6/6 + 5/5 + 1/1 = 12$ +
$\theta = 1$: $6/6 + 5/5 + 1/1 = 12$
$\theta = 0 + \theta = 1 = 12$

Determining flow values for loop:
Given $X_{1211} = \theta$ then flow at node 11 is "in more" and flow at node 12 is "in more".
Looking at flow along $X_{1206}$ must be "more" or "less".
Looking at flow at node 11 is "in less" and flow at node 12 is "in more".
Looking at flow along $X_{1206}$ must be "less" or "\theta - \theta".
So flow at node 11 is "less" so flow at node 16 is "in less".
Looking at flow along $X_{1206}$ must be "less" or "\theta - \theta".

Current flow:
$X_{1211} = 68 = \theta$, 68
$X_{1206} = 10 = \theta$
$X_{1205} = 8 = \text{trans}$
$X_{1204} = 10 = \text{trans}$
$X_{1202} = 8 = \text{trans}$
$X_{1207} = 8 = \text{trans}$
$X_{1201} = 6 = \text{trans}$
$X_{1206} = 8 = \text{trans}$
$X_{1211} = 8$
$X_{1206} = 8$
$X_{1201} = 8$

Iteration III - Step 1:
Find (x1 - x5) with (x1 = 0) (arbitrarily) using basic links and x1 = \( \theta + \phi \)

\[
\begin{align*}
\text{x1} - \text{x6} &= C_{1204} = 1; \text{x1} = -1 \\
\text{x1} - \text{x7} &= C_{1205} = 1; \text{x1} = 1 \\
\text{x1} - \text{x8} &= C_{1206} = 1; \text{x1} = 1 \\
\text{x1} - \text{x9} &= C_{1207} = 1; \text{x1} = 1 \\
\text{x1} - \text{x10} &= C_{1208} = 1; \text{x1} = 1 \\
\text{x1} - \text{x11} &= C_{1209} = 1; \text{x1} = 1 \\
\text{x1} - \text{x12} &= C_{1210} = 1; \text{x1} = 1 \\
\end{align*}
\]

\[\text{Need to break tie for X1205, X1206, X1207, X1208, X1209, X1210.}
\]

Need to break tie for X1205, X1206, X1207, X1208, X1209, and X1210 using original cost.

Need to break tie for X1205, X1206, X1207, and X1208 by plotting first occurrence.

Pick X1205 to add to basic group.
Step 3: Determine which basic variable to remove. Consider XHAB = 0 then XHAB u 18 = 0 and XHAB u E = 0 = 0.
Such that:
XHAB u i = 0
XHAB u i = 18 = 0 & i = 0, 1, 2, ..., 10
XHAB u E = 0 & i = 0
Choosing xHAB = 18 will satisfy all of the above and result in XHAB u 18 = 0, XHAB u 0, 1, and XHAB u E = 18 so xHAB will drop out of the basic group and XHAB will be reversed.
Recalculate cost:
\[ C_{base} = C_{init} + x_{HAB} * C_{cost} = C_{base} + x_{HAB} * C_{cost} \]
\[ C_{base} = C_{base} + x_{HAB} * C_{cost} \]

Determining flow values for loop:

Given XHAB = 0 then flow at node 64 is "in one" and flow at node 65 is "out one"

Looking at the flow along xHAB must be "zero" or "flow"

In this case the flow along xHAB is "out one" and flow at node 64 is "out zero"

Looking at the flow along xHAB is "out one" so flow at node 64 is "in zero" to flow along xHAB must be "less" or "<".

Current flow:
XHAB = 0, 65 = 60
XHAB = 12, 12 = 32
XHAB = 12, 6 = 30
XHAB = 18, 60 = 60
XHAB = 0, 6 = 30
XHAB = 18, 60 = 60
XHAB = 18, 60 = 60
XHAB = 0, 60 = 60
XHAB = 0, 60 = 60

Iteration IV - Step 1:
Find (u, v) with u < v (arbitrarily) using basis links and \( mi = v - u \)

\( u_0 = u_0 = u_0 = u_0 = u_0 = u_0 = u_0 = u_0 = u_0 = u_0 = u_0 \)
\( u_1 = u_1 = u_1 = u_1 = u_1 = u_1 = u_1 = u_1 = u_1 = u_1 = u_1 \)
\( u_2 = u_2 = u_2 = u_2 = u_2 = u_2 = u_2 = u_2 = u_2 = u_2 = u_2 \)
\( u_3 = u_3 = u_3 = u_3 = u_3 = u_3 = u_3 = u_3 = u_3 = u_3 = u_3 \)
\( u_4 = u_4 = u_4 = u_4 = u_4 = u_4 = u_4 = u_4 = u_4 = u_4 = u_4 \)
\( u_5 = u_5 = u_5 = u_5 = u_5 = u_5 = u_5 = u_5 = u_5 = u_5 = u_5 \)
\( u_6 = u_6 = u_6 = u_6 = u_6 = u_6 = u_6 = u_6 = u_6 = u_6 = u_6 \)
\( u_7 = u_7 = u_7 = u_7 = u_7 = u_7 = u_7 = u_7 = u_7 = u_7 = u_7 \)
\( u_8 = u_8 = u_8 = u_8 = u_8 = u_8 = u_8 = u_8 = u_8 = u_8 = u_8 \)
\( u_9 = u_9 = u_9 = u_9 = u_9 = u_9 = u_9 = u_9 = u_9 = u_9 = u_9 \)
\( u_{10} = u_{10} = u_{10} = u_{10} = u_{10} = u_{10} = u_{10} = u_{10} = u_{10} = u_{10} = u_{10} \)
\( u_{11} = u_{11} = u_{11} = u_{11} = u_{11} = u_{11} = u_{11} = u_{11} = u_{11} = u_{11} = u_{11} \)
\( u_{12} = u_{12} = u_{12} = u_{12} = u_{12} = u_{12} = u_{12} = u_{12} = u_{12} = u_{12} = u_{12} \)

Step 2: Determine which non-basic variables should be added to the basic group

XHAB = u_1 - u_0 = u_1 - u_2
XHAB = u_2 - u_1 = u_2 - u_3
XHAB = u_3 - u_2 = u_3 - u_4
XHAB = u_4 - u_3 = u_4 - u_5
XHAB = u_5 - u_4 = u_5 - u_6
XHAB = u_6 - u_5 = u_6 - u_7
XHAB = u_7 - u_6 = u_7 - u_8
XHAB = u_8 - u_7 = u_8 - u_9
XHAB = u_9 - u_8 = u_9 - u_{10}
XHAB = u_{10} - u_9 = u_{10} - u_{11}
XHAB = u_{11} - u_{10} = u_{11} - u_{12}

Need to break tie for XHAB, XHAB, XHAB, and XHAB using original cost value. Pick best cost.

XHAB = u_1 - u_0 = u_1 - u_2
XHAB = u_2 - u_1 = u_2 - u_3
XHAB = u_3 - u_2 = u_3 - u_4
XHAB = u_4 - u_3 = u_4 - u_5
XHAB = u_5 - u_4 = u_5 - u_6
XHAB = u_6 - u_5 = u_6 - u_7
XHAB = u_7 - u_6 = u_7 - u_8
XHAB = u_8 - u_7 = u_8 - u_9
XHAB = u_9 - u_8 = u_9 - u_{10}
XHAB = u_{10} - u_9 = u_{10} - u_{11}

Pick xHAB to add to basic group

XHAB = u_1 - u_0 = u_1 - u_2
XHAB = u_2 - u_1 = u_2 - u_3
XHAB = u_3 - u_2 = u_3 - u_4
XHAB = u_4 - u_3 = u_4 - u_5
XHAB = u_5 - u_4 = u_5 - u_6
XHAB = u_6 - u_5 = u_6 - u_7
XHAB = u_7 - u_6 = u_7 - u_8
XHAB = u_8 - u_7 = u_8 - u_9
XHAB = u_9 - u_8 = u_9 - u_{10}
XHAB = u_{10} - u_9 = u_{10} - u_{11}
Step 3:
Determine with basic variable to remove. Consider X980 = 0 then X980 + x = 0 and X980 = x - 0.
Such that:
X980 + x = 0
X980 = 0
x = 0
Choosing x = 0 will satisfy all of the above and result in X980 = 0, X980 = 0, and X980 = x - 0.

Reinitialize cost:
- CB950*1000 = CB950*990000 + CB950*980000 + CB950*970000 + CB950*960000 + CB950*950000 + CB950*940000 + CB950*930000 + CB950*920000 + CB950*910000 + CB950*900000 + CB950*890000 + CB950*880000 + CB950*870000 + CB950*860000 + CB950*850000 + CB950*840000 + CB950*830000 + CB950*820000 + CB950*810000 + CB950*800000 + CB950*790000 + CB950*780000 + CB950*770000 + CB950*760000 + CB950*750000 + CB950*740000 + CB950*730000 + CB950*720000 + CB950*710000 + CB950*700000 + CB950*690000 + CB950*680000 + CB950*670000 + CB950*660000 + CB950*650000 + CB950*640000 + CB950*630000 + CB950*620000 + CB950*610000 + CB950*600000 + CB950*590000 + CB950*580000 + CB950*570000 + CB950*560000 + CB950*550000 + CB950*540000 + CB950*530000 + CB950*520000 + CB950*510000 + CB950*500000 + CB950*490000 + CB950*480000 + CB950*470000 + CB950*460000 + CB950*450000 + CB950*440000 + CB950*430000 + CB950*420000 + CB950*410000 + CB950*400000 + CB950*390000 + CB950*380000 + CB950*370000 + CB950*360000 + CB950*350000 + CB950*340000 + CB950*330000 + CB950*320000 + CB950*310000 + CB950*300000 + CB950*290000 + CB950*280000 + CB950*270000 + CB950*260000 + CB950*250000 + CB950*240000 + CB950*230000 + CB950*220000 + CB950*210000 + CB950*200000 + CB950*190000 + CB950*180000 + CB950*170000 + CB950*160000 + CB950*150000 + CB950*140000 + CB950*130000 + CB950*120000 + CB950*110000 + CB950*100000 + CB950*900000 + CB950*800000 + CB950*700000 + CB950*600000 + CB950*500000 + CB950*400000 + CB950*300000 + CB950*200000 + CB950*100000 + CB950*000000

Determining flow values for loop:
- Given X980 = x if flow out at node 89 is "in more" and flow out at node 89 is "in more" so flow along X980 must be "less" or "up".
- Looking at equation X980 = x + w9 + x - 0 + x - 0 + x = 0, the flow at node 89 is "out less" so flow along X980 must be "less" or "up".
- Looking at X980 we see flow at node 89 is "out less" so flow at node 89 is "in less" so flow along X980 must be "less" or "up".

Current flow:
X980 = 0
X660 = 12 - x
X692 = 10
X650 = 10
X660 = 0 + x
X670 = 10 - x
X627 = 10 - x
X650 = 6
X611 = 63
X610 = 63 - x
X630 = 63 - x

Step 2:
Determine which non-basic variable should be added to the basic group.

Iteration V - Step 1:
Find (x + w) with x = 0 (arbitrarily) using basic links and w = x-0

Need to break ties for X660, X692 and X690.

Need to break ties for X660, X690, and X692 by plotting first occurrence.

Pick X690 to add to basic group.
Step 3: Determine which basic variable to remove. Consider XN04 = x then XN04 = 10 - x, XN06 = 10 - y, and YN02 = y - z.
Such that:
XN04 = 10 - x
XN06 = 10 - y
YN02 = y

Choosing x = 10 will satisfy all of the above and result in XN04 = 10, XN06 = 0, XN08 = 0, and YN02 = 10 so XN04 and XN06 will drop out of the basic group and YN02 will be reversed.

Recalculate flow:

Determining flow values for loop:

Given XN04 = 10 then flow at node 10 is "in loop" and flow at node 06 is "in loop".
Looking at flow along XN06 must be "less" or "= -q"
Looking at flow along XN04 must be "less" or "= -q"
Looking at XN04 we see flow at node 06 is "out less" so flow at node 02 is "in less".
looking at flow along XN06 must be "less" or "= -q"
Looking at flow along XN02 must be "less" or "= -q"
Looking at flow along XN06 must be "less" or "= -q".
Step 3:
Determine which basic variable to remove. Consider $X_{1297} = 0$ then $X_{1297} = 5 - B_8$.

Such that:

$X_{1297} = 0$ if $B_8 = 0$ or $B_8 = 5$

Choosing $B_8 = 5$ will satisfy all of the above and result in $X_{1297} = 5$, $X_{1297} = 0$,
and $X_{1297} = -5$. So $X_{1297}$ will be dropped out of the basic group and $X_{1297}$ will be reserved.

Recalculate cost:

\[ C_{05939} = C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 + C_{05939}B_0 \]

Determining flow values for loop:

Given $X_{1297} = 0$ then flow at node 12 is "not less" and flow at node 07 is "in less".

To flow along $X_{1297}$ must be "less" or "=cf".

Linking along $X_{1297}$ as "less" and flow at node 06 is "not less".

So flow along $X_{1297}$ must be "less" or "=cf".

Linking at $X_{1297}$ we see flow at node 12 is "not less" so flow at node 06 is "in less"

So flow along $X_{1297}$ must be "less" or "=cf".

Current flow:

$X_{1080} = 60 - 31$
$X_{1040} = 12 - 32$
$X_{0900} = 8$
$X_{0900} = 10$
$X_{0900} = 10$
$X_{0900} = 10$
$X_{0900} = 30$
$X_{0900} = 10$
$X_{1100} = 10$
$X_{1100} = 10$
$X_{1100} = 10$
$X_{1100} = 10$
$X_{1100} = 10$

Step 1:
Determine which non-basic variables should be added to the basic group.

$X_{1080} = w_1 - w_4 - C_{3240} + 1$
$X_{1090} = w_1 - w_4 - C_{3240} + 1$
Need to break ties for $X_{1080}$, $X_{1090}$, and $X_{1100}$ using original cost value. Pick least cost.
$X_{1080} = 3$

$X_{1100} = 3$
$X_{1100} = 3$

Pick $X_{1100}$ to add to basic group.

Iteration VII + Step 3:
Find $s_{i,j}$ with $s_{i,j} = 0$ (arbitrarily) using basic links and $\alpha_i - \alpha_j$.
Step 3:
Determine which basic variable to remove. Consider $X_{132} = 0$ then $X_{124} = 0 - 0$.
and $X_{134} = 12 - 0$.
Such that:
1. $X_{132} = 0$, $X_{124} = 0 - 0$ or $0 - 0$.
2. $X_{134} = 12 - 0$ or $0 - 0$.
Choosing such $x$ will satisfy all of the above and result in $X_{132} = 0$, $X_{124} = 0$, and $X_{134} = 12$ so $X_{134}$ will drop out of the basis group.

Recalculating cost:

Determining flow values for loop:

Given $X_{132} = 0$ then flow at node 12 is "in wise" and flow at node 02 is "in wise".
Looking at flow along $X_{132}$ must be "up" or "up".
Looking at flow along $X_{124}$ must be "less" or "down".
Looking at $X_{134}$ we see flow at node 13 is "in wise" so flow at node 04 is "in wise".

Current Flow:

Need to break ties for $X_{109}$, $X_{112}$, and $X_{134}$ using original cost value. Pick least cost.

Pick $X_{134}$ to add to basis group.
Step 1:
Determine which basic variable to remove. Consider $X_{123} = 0$ then $X_{132} = 6 - 0$.

$X_{132} = 6 - 0 = 6$ or $6 = 6$

$X_{127} = 5 - 0 = 5$ or $5 = 5$

$X_{128} = 5 - 0 = 5$ or $5 = 5$

$X_{129} = 5 - 0 = 5$ or $5 = 5$

$X_{132} = 6 - 0 = 6$ or $6 = 6$

Choosing $m = 5$ will satisfy all of the above and result in $X_{123} = 5$, $X_{132} = 5$, $X_{127} = 5$, $X_{128} = 5$, $X_{129} = 5$, and $X_{132} = 6$ so $X_{132}$ will drop out of the basic group.

Recalculate cost:

<table>
<thead>
<tr>
<th>$X_{123}$</th>
<th>$X_{127}$</th>
<th>$X_{128}$</th>
<th>$X_{129}$</th>
<th>$X_{132}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$400$</td>
<td>$300$</td>
<td>$300$</td>
<td>$300$</td>
<td>$500$</td>
</tr>
</tbody>
</table>

Determining flow values for loop:

Given $X_{123} = 5$ then flow at node 10 is "not zero" and flow at node 13 is "in zero".

Looking at flow along $X_{123}$ must be "new" or "none".

Looking at flow along node 10 is "in zero" so flow at node 13 is "in zero".

Looking at flow along $X_{132}$ must be "new" or "none".

Looking at flow along node 10 is "in zero" so flow at node 13 is "in zero".

Current Flow:

<table>
<thead>
<tr>
<th>Node</th>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>123</td>
<td>5</td>
</tr>
<tr>
<td>127</td>
<td>5</td>
</tr>
<tr>
<td>128</td>
<td>5</td>
</tr>
<tr>
<td>129</td>
<td>5</td>
</tr>
<tr>
<td>132</td>
<td>6</td>
</tr>
</tbody>
</table>

Step 2:
Determine which non-basic variable should be added to the basic group

Need to break ties for $X_{132}$ and $X_{123}$ using original cost value. Pick least cost.

Pick $X_{132}$ to add to basic group.
Step 3:
Determine which basic variable to remove. Consider $X_{193}$ is $b$ then $X_{193} = 5 + b$, $X_{192} = 5 + b$, $X_{190} = 5 + b$, and $X_{191} = 5 + b$.
Such that:
$X_{190} = 5 + a = 0$
$X_{191} = 5 + a = 0$
$X_{192} = 5 + a = 0$
$X_{193} = 5 + b = 0$
$X_{200} = 3 - b = 0$ or $b = 3$
$X_{201} = 18 - b = 0$ or $b = 18$
Choosing map $b$ will satisfy all of the above and result in $X_{190} = 5$, $X_{191} = 0$, $X_{192} = 0$, $X_{193} = 5$, and $X_{200} = 7$. So $X_{200}$ will drop out of the basic group.

Recalculate cost:

Choosing map $b$ results in the following cost:

Determining flow values for loop:

Given $X_{190} = 5$ then flow at node 05 is "out none" and flow at node 19 is "in none"
Looking at flow along $X_{193}$ must be "more" or "less".
Looking at flow along $X_{192}$ is "more" or "less" so flow at node 19 is "in none".
Looking at flow along $X_{191}$ must be "more" or "less".
Looking at flow along $X_{200}$ must be "less" or "more" so flow at node 06 is "in less".
Looking at flow along $X_{200}$ must be "less" or "more" so flow at node 06 is "in less".
So flow along $X_{200}$ must be "less" or "more".

Current flow:

Step 2:
Determine which non-basic variables should be added to the basic group.

Iteration 5 = Step 1:
Find set $S$ with $S = (5, 6, 7)$ (arbitrarily) using basic links and $S = \{5, 6, 7\}$

$w_1 = w_2 = 0$ in $S$; $w_3 = 1$

$w_0 = X_{200} = 0$

$x_0 = X_{200} = 0$

$x_5 = X_{190} = 5 + b$

$x_6 = X_{191} = 0$

$x_7 = X_{192} = 0$

$x_8 = X_{193} = 5 + b$

Need to break tie for X1200 and X1201 using original cost value. Pick least cost.

PICK X1200 to add to basic group.
Step 1:
Determine which basic variable to remove. Consider X1006 ≥ 7 then X1006 ≥ 16 - 8, X1006 ≥ 7 - 6, X1018 ≥ 3 + 2, and X1018 ≥ 8 + 0.

Such that:
X1006 ≥ 8 ≥ 7 ≥ 0
X1006 is 16 - 8 ≥ 0 or 0 ≥ 7
X1006 is 7 - 6 ≥ 0 or 0 ≥ 7
X1018 is 3 + 2 ≥ 0
X1018 is 8 + 0 ≥ 0

Choosing max ≥ 7 will satisfy all of the above and result in X1006 ≥ 7, X1006 ≥ 9, X1006 ≥ 7, X1006 ≥ 6, and X1018 ≤ 15 so X1006 will drop out of the basic group.

Recalculate cost:

Determine flow values for loop;

Given X1006 ≥ 7 then flow at node 12 is “less or ’’>’’ and flow at node 08 is “in” moral
Looking at X1018 we see flow at node 13 is “less or ”>’’ so flow at node 10 is “less or ”>’’

Looking at X1018 we see flow at node 10 is “less or ”>’’

Looking at X1018 we see flow at node 10 is “less or ”>’’

Looking at node 08 we see flow at node 06 is “less or ”>’’

Looking at node 08 we see flow at node 06 is “less or ”>’’ so flow at node 06 is “less or ”>’’

Looking at node 08 we see flow at node 06 is “less or ”>’’ so flow at node 06 is “less or ”>’’

Looking at node 08 we see flow at node 06 is “less or ”>’’ so flow at node 06 is “less or ”>’’

Step 2:
Determine which non-basic variable should be added to the basic group.

Iteration XI - Step 1:
Find (xj - aj) with xj ≤ a j (arbitrarily) using basic links and xj - aj ≥ 0.

v001 - v004 + v007 = 3, v005 = 1
v001 - v007 + v008 = 1, v007 - (-1) = 0, v007 = 1
v001 - v007 + v008 = 1, v007 - (-1) = 0, v007 = 1
v001 - v005 + v007 = 1, v007 - (-1) = 0, v007 = 1
v001 - v005 + v007 = 1, v007 - (-1) = 0, v007 = 1
v001 - v005 + v007 = 1, v007 - (-1) = 0, v007 = 1
v001 - v005 + v007 = 1, v007 - (-1) = 0, v007 = 1
v001 - v005 + v007 = 1, v007 - (-1) = 0, v007 = 1

Need to bring in link to solve for v008: cheapest incoming
v001 - v007 + v008 = 0, (-1) - v007 = 0, v007 = 0

Current flow:
X0036 = 69 + 61
X0037 = 60
X0038 = 10
X1000 = 10
X1006 = 69
X1007 = 8 = trans
X1011 = 9
X1012 = 12
X1013 = 9

Need to break ties for X1004 and X1012 using original cost value. Pick least cost.

v1004 = 5
X1012 = 3

Pick X1004 to add to basic group.
Step 3:
Determine which basic variable to remove. Consider X1264 = 0 then X416 = 18 + \( \delta \), X518 = 18 + \( \delta \), X1218 = 15 + \( \delta \), X136 = 7 + \( \delta \), X518 = 0 + \( \delta \), and X136 = 0 + \( \delta \).
Such that:
X1218 = 0 = \( \beta \)
X518 = 18 + \( \delta \) = \( \beta \)
X518 = 18 + \( \delta \) = \( \beta \)
X136 = 15 + \( \delta \) = \( \beta \)
X1264 = 7 + \( \delta \) = \( \beta \)
X1218 = 0 + \( \delta \) = \( \beta \)
X518 = 0 + \( \delta \) = \( \beta \)
X136 = 0 + \( \delta \) = \( \beta \)
Choosing \( \delta = 0 \) will satisfy all of these above and result in X1264 = 0, X518 = 18, X518 = 18, X136 = 15, X1264 = 7, X518 = 0, and X136 = 0 so X1264 and X518 will drop out of the basic group.

Recalculate cost:
\[
\begin{align*}
0^* \text{(set of nodes)} & = \{250 + 1250 + 150 + 750 + 970 + 500 + 1250 + 1750 + 1000 + 1250 \}
\]
Determining flow values for loop:

Given \( \delta = 0 \) then flow at node 6 is “out more” and flow at node 8 is “in more” so flow along X1264 must be “more” or “\( \delta \)”. Flow along X1218 must be “more” or “\( \delta \)”.

Looking at X1218 we see flow at node 10 is “in more” so flow at node 8 is “out more”. Flow along X518 must be “more” or “\( \delta \)”.

Looking at X518 we see flow at node 10 is “in more” so flow at node 13 is “in more”. Flow along X1218 must be “more” or “\( \delta \)”.

Looking at X1218 we see flow at node 10 is “out more” so flow at node 8 is “in more”.

Looking at X136 we see flow at node 13 is “out more” so flow at node 6 is “in less”. Flow along X518 must be “less” or “\( -\delta \)”.

Looking at X518 we see flow at node 10 is “out less” so flow at node 8 is “in less”. Flow along X1218 must be “less” or “\( -\delta \)”.

COST = 0; STOP PHASE 1

**Current flow**

<table>
<thead>
<tr>
<th>Node</th>
<th>FlowIn</th>
<th>FlowOut</th>
<th>Supply</th>
<th>Demand</th>
<th>Balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>02</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>-12</td>
<td>Yes</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>06</td>
<td>9</td>
<td>10</td>
<td>19</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>07</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>09</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>-16</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>19</td>
<td>6</td>
<td>(1x24)</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>24</td>
<td>4</td>
<td>(12x15)</td>
<td>0</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Iteration I - Step 1: Find (v4 - v5) with v4 = 0 (first node with most links = 3) using basic links, original costs, and v4-v5 = 0.

1. v4 = v5 = C0284 = 1; v5 - 9 = 1; v4 = 10
2. v4 = v5 = C0389 = 1; v5 - 6 = 1; v4 = 9
3. v4 = v5 = C0310 = 4; v5 - 4 = 4
4. v4 = v5 = C1203 + 1; v5 - 2 = 1
5. v4 = v5 = C2213 + 4; v5 - 4 = 4
6. v4 = v5 = C1302 + 5; v5 - 5 = 5; v4 = 5
7. v4 = v5 = C1009 + 4; v5 - 4 = 4; v4 = 6
8. v4 = v5 = C1000 + 5; v5 - 5 = 5; v4 = 7

Need to bring in nodes to solve for v4, v5, v6, and v7 (cheapest incoming)

v4 = v5 = C0003 = 4; v6 = 5; v7 = 6
v5 = v6 = C0006 = 1; v6 = 6; v8 = 1
v6 = v7 = C1007 + 5; v7 = 11
v7 = v8 = C1300 + 2; v8 = 9; v9 = 9

Phase II - Find the optimal solution from the network produced at the end of Phase I.

Starting cost = C0138*Y0004 + C0309*Y0305 + C0132*X0132 + C1013*X1013 + C1213*Y1213 + C1004*Y1004 + C1005*Y1005 = 7 + 3 + 2 + 4 + 21 + 3 + 3 + 3 + 3 = 48

Step 2: Determine which non-basic variable should be added to the basic group.

X9112 = v6 - v7 = 1 - 0 = 1
X0100 = v7 - v8 = 1 - 0 = 1
X0101 = v8 - v9 = 1 - 0 = 1
X0102 = v9 = 1

Pick the largest positive value, X9112 = 1, to add to basic group

Step 3: Determine which basic variable to remove. Consider X9112 = 0 then X9112 = 1.

X9112 = v6 - v7 = 0
X0100 = v7 - v8 = 0
X0101 = v8 - v9 = 0
X0102 = v9 = 0

Choosing v6 = 1 will satisfy all of the above and result in X9112 = 1, X9112 = 0, X0100 = 0, X0101 = 0, and X0102 = 0 so X9112 will drop out of the basic group.

Recalculate cost
C0138*Y0004 + C0309*Y0305 + C0132*X0132 + C1013*X1013 + C1213*Y1213 + C1004*Y1004 + C1005*Y1005 = 7 + 3 + 2 + 4 + 21 + 3 + 3 + 3 + 3 = 48

Determining flow values for loop:

Given X9112 = 0 then flow at node 6 is "out more" and flow at node 11 is "in less" so flow along X9112 must be "more" or "up".
Iteration II - Step 1:
First (vi, xvi) with xvi = 0 (first node with most links = 1) using basic
lines, original costs, and xvi = xvi + 1

Step 2:
Determine which non-basic variable should be added
to the basic group

Pick the largest positive value, x2000 = 5,
to add to basic group

Step 3:
Determine which basic variable to remove. Consider X2000 = 0 then X2000 = 0 + 0,
X2010 = 10 - 0, X2015 = 24 - 0, and X3000 = 16 - 0.

X2010 is 10 or 0 or 10 > 0
X2015 is 24 or 0 or 24 > 0
X3000 is 16 or 0 or 16 > 0

Choosing x2015 = 15 will satisfy all of the above and result in X2000 = 15, X2010 = 0, X2015 = 3, and X3000 = 0 so X3000 will dropped out of the basic group.

Recalculate cost

Determining flow values for loop:
Given X3000 as 0 then flow at node 00 is "not more" and flow at node 08 is "in more" so flow along X3000 must be "more" or "up"

Looking at X3000 we see flow at node 08 is "in less" and flow along node 13 is "out less" so flow along X3000 must be "less" or "down"

Looking at X3000 we see flow at node 13 is "out less" so flow along node 08 is "in less" so flow along X3000 must be "less" or "up"

Looking at X3000 we see flow at node 08 is "in more" so flow at node 00 is "in more" so flow along X3000 must be "more" or "up"
ATTACHMENT C – CHAYEN SOLUTIONS BY-HAND

The following MS PowerPoint presentations detail the by-hand solutions for some of the Chayen algorithm examples called out in this work. Because Phase 2 using the Linprog() function is not reproducible by-hand,

Phase 2 in Examples 1 and 2 use the conventional MCF method.
C.1. Chayen By-Hand Solution for Chapter V, Example #1

Link to MS PowerPoint Presentation ➔ https://tinyurl.com/2pms48cw

Example #1 (9-Node/14 Link) Network — Chayen Phase 1 / MCF Phase 2

The original network is used to generate the “Rank/Park” matrix. Then we iterate through the matrix until all the supply/demand is distributed. This will result in the Phase 1 feasible solution.

<table>
<thead>
<tr>
<th>Rank</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>(3,9)</td>
<td>(7,9)</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>8+8=16</td>
</tr>
</tbody>
</table>

Matrix[1] 3 7

1 1 0 3
9 1 1 5 3 2
5 3 0 5+5=10

Matrix[2] 3 7

1 1 0 3
9 1 1 5 3 2
5 3 0 5+5=10

Matrix[3] 3 7

1 1 0 3
9 1 1 5 3 2
5 3 0 5+5=10

Phase II — Find the optimal solution from the network produced at the end of Phase I.
Starting cost = C[0]*x0 + C[1]*x1 + C[2]*x2 + C[3]*x3 + C[4]*x4 + C[5]*x5 + C[6]*x6 + C[7]*x7 + C[8]*x8 + C[9]*x9 + C[10]*x10 + C[11]*x11 + C[12]*x12 + C[13]*x13 + C[14]*x14

Iteration I — Step 1:
Find (ux - wv) with ux = 0 (first node with most links = 3) using basic links, original costs, and ux = wv = 0.
x3 = x5 + C[7] = 6; w3 = 6
x5 = x6 + C[8] = 6; w5 = 6
w7 = w8 = C[9]= 2; w7 = (-2) = 2
Need to bring in links to solve for w7, w8, and w9 (Cheapest Incoming)
x7 = w7 = 2; (-6) = w7 = 2; w9 = -8
x5 = w5 = 1; (-4) = w5 = 1; w6 = -7
w9 = w9 = 1; (-4) = w9 = 1; w6 = -3

Step 2:
Determine which non-basic variable should be added to the basic group.

| X23: | x2 - x3 = (-6) = 0 - 10 |
| X25: | x2 - x5 = (-6) = 0 - 10 |
| X47: | x4 - x7 = (-7) = 5 - 12 |
| X58: | x5 - x8 = (-5) = 6 - 11 |
| X78: | x7 - x8 = (-2) = 0 - 8 |

No positive values exist; stop Phase 2.
112

Current flow, Original cost
X35 = 5  
X54 = 6  Flow*Cost = 30
X51 = 3  
X51 = 6  Flow*Cost = 18
X85 = 2  
X85 = 6  Flow*Cost = 8
X78 = 3  
X78 = 3  Flow*Cost = 9
X89 = 3  
X89 = 2  Flow*Cost = 6

----------
FreeSlack Value = 71

<table>
<thead>
<tr>
<th>Mode</th>
<th>FlowIn</th>
<th>Supply</th>
<th>FlowOut</th>
<th>Demand</th>
<th>Balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>Yes</td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>07</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>09</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3
C.2. Chayen By-Hand Solution for Chapter V, Example #2

Link to MS PowerPoint Presentation  →  https://tinyurl.com/n9w567jp

Example #2 [13-Node/20 Link] Network – Chayen Phase 1 / MCF Phase 2

The original network is used to generate the “Rank/Path” matrix. Then we iterate through the matrices until all the Supply/Demand is distributed. This will result in the Phase 1 feasible solution.

Matrix[1] 1 4 10 13 Demand
2 [ ] [ ] [ ] [ ] [ ] 12 4=6
8 [ ] [ ] [ ] [ ] [ ] 16
11 [ ] [ ] [ ] [ ] [ ] 1
Supply 9 10 6 4 29+29=58

Matrix[2] 1 4 10 13 Demand
2 [ ] [ ] [ ] [ ] [ ] 8
8 [ ] [ ] [ ] [ ] [ ] 16
11 [ ] [ ] [ ] [ ] [ ] 2=1=0
Supply 9 10 6=5=0 24+24=48

Matrix[3] 1 4 10 13 Demand
2 [ ] [ ] [ ] [ ] [ ] 8=5=3
8 [ ] [ ] [ ] [ ] [ ] 16
11 [ ] [ ] [ ] [ ] [ ] 0
Supply 9 10 5=0=0 19+10=38

Matrix[4] 1 4 10 13 Demand
2 [ ] [ ] [ ] [ ] [ ] 3
8 [ ] [ ] [ ] [ ] 10=6=0
11 [ ] [ ] [ ] [ ] [ ] 0
Supply 9 10=0=0 9+9=18

Matrix[5] 1 4 10 13 Demand
2 [ ] [ ] [ ] [ ] [ ] 3
8 [ ] [ ] [ ] [ ] 6=0
11 [ ] [ ] [ ] [ ] [ ] 0
Supply 9=6=3 0 0 6=6=12

Matrix[6] 1 4 10 13 Demand
2 [ ] [ ] [ ] [ ] [ ] 3=3=0
8 [ ] [ ] [ ] [ ] [ ] 0
11 [ ] [ ] [ ] [ ] [ ] 0
Supply 3=3=0 0 0 0=1=0
STOP
Phase II - Find the optimal solution from the network produced at the end of Phase I.

Starting cost = 0.5848x1000 + 0.1005x1000 + 0.0484x1000 + 0.0045x1000 + 0.0035x1000 + 0.0021x1000 + 0.0002x1000

Step 2: Determine which non-basic variable should be added to the basic group

Pick the largest positive value, X112 = 8, to add to basic group

Step 3: Determine which basic variable to remove. Consider X112 = 0 then X108 = 9 - 8 = 1, X201 = 19 - 8 = 11, and X202 = 1 - 8 = -7.

X112 is 0 = 0

X108 is 19 - 8 = 11 > 0 and X201 = 11 > 0

X202 is 1 - 8 = -7 < 0

Choosing mex p = 1 will satisfy all of the above and result in X112 = 1, X108 = 8, X201 = 18, X202 = 2, and X202 = 0 to X202 will dropped out of the basic group.

Reallocate cost

Determining flow values for loop:

Given X112 as 0 then Flow at node 01 is "out more" and flow at node 31 is "in more" so flow along X112 must be "more" or "yf".

Looking at X018 we see Flow at node 01 is "out less" and Flow at node 04 is "in less" so flow along X112 must be "less" or "yf".

Looking at X048 we see Flow at node 04 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".

Looking at X112 we see Flow at node 18 is "out less" so flow along X112 must be "less" or "yf".
C.3. Chayen By-Hand Solution for Chapter V, Example #5 – Phase 1

Link to MS PowerPoint Presentation → https://tinyurl.com/uu35em55

The original network used to generate the Rank/Path matrix:

Resulting feasible solution after Chayen Phase 1:

<table>
<thead>
<tr>
<th>Rank/Path</th>
<th>Node 1</th>
<th>Node 4</th>
<th>Node 9</th>
<th>Node 13</th>
<th>Node 18</th>
<th>Node 19</th>
<th>Node 22</th>
<th>Node 26</th>
<th>Node 31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 2</td>
<td>11</td>
<td>6</td>
<td>13</td>
<td>11</td>
<td>4</td>
<td>16</td>
<td>17</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>Node 5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>12</td>
<td>5</td>
<td>15</td>
<td>18</td>
<td>5</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>Node 17</td>
<td>9</td>
<td>14</td>
<td>9</td>
<td>15</td>
<td>17</td>
<td>27</td>
<td>11</td>
<td>9</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>Node 20</td>
<td>11</td>
<td>29</td>
<td>11</td>
<td>15</td>
<td>5</td>
<td>18</td>
<td>28</td>
<td>12</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>Node 23</td>
<td>25</td>
<td>17</td>
<td>6</td>
<td>15</td>
<td>7</td>
<td>18</td>
<td>25</td>
<td>12</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>Node 24</td>
<td>11</td>
<td>14</td>
<td>1</td>
<td>14</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>9</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>Node 30</td>
<td>22</td>
<td>33</td>
<td>22</td>
<td>24</td>
<td>12</td>
<td>73</td>
<td>18</td>
<td>19</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>96</td>
<td></td>
</tr>
</tbody>
</table>

Begin with an empty iteration matrix.

Using the Rank/Path matrix, select the cheapest location. In this case, the intersection of Demand node #24 and Supply node #9 has a cost of 1 and is selected as the pivot point. Initially, supply is 8 and demand is 5. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

Supply 1 2 8 6 5 10 1 6 9 96

Demand 1 4 9 13 18 19 22 26 31

<table>
<thead>
<tr>
<th>[0]</th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>8</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
</tbody>
</table>

Supply 1 2 8 5 10 1 6 9 96

<table>
<thead>
<tr>
<th>[1]</th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>8</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>5-5=0</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
</tbody>
</table>

Supply 1 2 8 5 10 1 6 9 96

Demand 96-10=86
For iteration #2: use the Rank/Path matrix to select the next cheapest location. In this case, the intersection of Demand node #2 and Supply node #18 has a cost of 4 and is selected as the pivot point. Initially, supply is 5 and demand is 18. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[5]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>18-5=13</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>8</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>5-5=0</td>
<td>10</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>86-10=76</td>
</tr>
</tbody>
</table>

For iteration #3: use the Rank/Path matrix to select the next cheapest location. At this point there are five locations with a cost of 5. Select the row with the most demand and then, if there is a tie, select the column with the most supply. In this case, the intersection of Demand node #2 and Supply node #26 has the largest demand and a cost of 5; this is selected as the pivot point. Initially, supply is 6 and demand is 13. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[5]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>8</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>76-12=64</td>
</tr>
</tbody>
</table>

For iteration #4: use the Rank/Path matrix to select the next cheapest location. At this point there are three locations with a cost of 5. Select the row with the most demand and then, if there is a tie, select the column with the most supply. In this case, the intersection of Demand node #5 and Supply node #19 has the largest demand and a cost of 5; this is selected as the pivot point. Initially, supply is 9 and demand is 6. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[5]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[3]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>8-3=5</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>64-6=58</td>
</tr>
</tbody>
</table>

For iteration #5: use the Rank/Path matrix, select the cheapest location. In this case, the intersection of Demand node #17 and Supply node #13 has a cost of 4 and is selected as the pivot point. Initially, supply is 9 and demand is 1. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[5]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[3]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>1-1=0</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[0]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>9-1=8</td>
<td>58-2=56</td>
</tr>
</tbody>
</table>

117
For iteration #6, use the Rank/Path matrix to select the next cheapest location. In this case, the intersection of Demand node #13 and Supply node #31 costs 8 and is selected as the pivot point. Initially, supply is 8 and demand is 7. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8-7=1</td>
</tr>
</tbody>
</table>

For iteration #7, use the Rank/Path matrix, select the cheapest location. In this case, the intersection of Demand node #2 and Supply node #4 has a cost of 9 and is selected as the pivot point. Initially, supply is 2 and demand is 13. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>13-2=11</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8-7=1</td>
</tr>
</tbody>
</table>

For iteration #8, use the Rank/Path matrix to select the next cheapest location. In this case, the intersection of Demand node #1 and Supply node #2 has a cost of 10 and is selected as the pivot point. Initially, supply is 1 and demand is 11. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>5-1=4</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2-1=1</td>
</tr>
</tbody>
</table>

For iteration #9, use the Rank/Path matrix, select the cheapest location. At this point there are two locations with a cost of 11. They are in the same demand row so we break the tie by selecting the greatest available supply. In this case, the intersection of Demand node #20 and Supply node #13 has a cost of 11 and is selected as the pivot point. Initially, supply is 6 and demand is 2. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For iteration #10, use the Rank/Path matrix, select the cheapest location. At this point there are two locations with a cost of 11. They are in the same demand row so we break the tie by selecting the greatest available supply. In this case, the intersection of Demand node #20 and Supply node #13 has a cost of 11 and is selected as the pivot point. Initially, supply is 6 and demand is 2. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For iteration #11, use the Rank/Path matrix, select the cheapest location. At this point there are two locations with a cost of 11. They are in the same demand row so we break the tie by selecting the greatest available supply. In this case, the intersection of Demand node #20 and Supply node #13 has a cost of 11 and is selected as the pivot point. Initially, supply is 6 and demand is 2. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>1</td>
</tr>
<tr>
<td>Supply</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For iteration #12, use the Rank/Path matrix, select the cheapest location. At this point there are two locations with a cost of 11. They are in the same demand row so we break the tie by selecting the greatest available supply. In this case, the intersection of Demand node #20 and Supply node #13 has a cost of 11 and is selected as the pivot point. Initially, supply is 6 and demand is 2. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.
For iteration #10, use the Rank/Path matrix to select the next cheapest location. At this point, there are two locations with a cost of 12. Select the greatest demand. In this case, the intersection of Demand node 9 and Supply node 13 has a cost of 12 and is selected as the pivot point. Initially, supply is 4 and demand is 11. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[4]</td>
<td>[5]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>11-4=7</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td>[0]</td>
<td>[3]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[6]</td>
<td>[7]</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>[9]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>1</td>
</tr>
</tbody>
</table>

Supply 0 0 0 4-4=0 0 10 1 0 1 32-8=24

For iteration #11, use the Rank/Path matrix, select the cheapest location. In this case, the intersection of Demand node 30 and Supply node 19 has a cost of 12 and is selected as the pivot point. Initially, supply is 10 and demand is 1. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[4]</td>
<td>[5]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td>[0]</td>
<td>[3]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[6]</td>
<td>[7]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>1-1=0</td>
</tr>
</tbody>
</table>

Supply 0 0 0 0 0 10-1=9 1 0 1 26-2=22

For iteration #12, use the Rank/Path matrix to select the next cheapest location. In this case, the intersection of Demand node 9 and Supply node 13 has a cost of 16 and is selected as the pivot point. Initially, supply is 1 and demand is 7. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[4]</td>
<td>[5]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>7-1=6</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td>[0]</td>
<td>[3]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[6]</td>
<td>[7]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
</tbody>
</table>

Supply 0 0 0 0 0 0 1-1=0 1 0 22-2=20

For iteration #13, use the Rank/Path matrix, select the cheapest location. In this case, the intersection of Demand node 9 and Supply node 18 has a cost of 16 and is selected as the pivot point. Initially, supply is 1 and demand is 6. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[4]</td>
<td>[5]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>6-1=5</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td>[0]</td>
<td>[3]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[6]</td>
<td>[7]</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
</tbody>
</table>

Supply 0 0 0 0 0 9 1-1=0 0 0 20-2=18
For iteration #14, use the Rank/Path matrix to select the next cheapest location. At this point, there are two locations with a cost of 13. Select the one with the greatest demand. In this case, the intersection of Demand node #2 and Supply node #15 has a cost of 13 and is selected as the pivot point. Initially, supply is 9 and demand is 5. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

<table>
<thead>
<tr>
<th>Rank/Path</th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[4]</td>
<td>[5]</td>
<td>[5]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>5-5=0</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td>[0]</td>
<td>[3]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[6]</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[9]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>Supply</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9-5=4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18-10=8</td>
<td></td>
</tr>
</tbody>
</table>

For iteration #15, use the Rank/Path matrix, select the cheapest location. In this case, the intersection of Demand node #4 and Supply node #19 has a cost of 4 and is selected as the pivot point. Initially, supply is 9 and demand is 4. Move the least of these values to the pivot point and reduce both the supply and demand values by the amount moved.

This results in all supply and all demand distributed.

<table>
<thead>
<tr>
<th>Rank/Path</th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>13</th>
<th>18</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>31</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[4]</td>
<td>[5]</td>
<td>[5]</td>
<td>[1]</td>
<td>[0]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td>[0]</td>
<td>[3]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>4-4=0</td>
</tr>
<tr>
<td>17</td>
<td>[0]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[2]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[6]</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>[0]</td>
<td>[0]</td>
<td>[5]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[9]</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>0</td>
</tr>
<tr>
<td>Supply</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4-4=0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8-8=0</td>
<td></td>
</tr>
</tbody>
</table>

The cost matrix is generated by calculating the summation of the dot product of the solution matrix[15] and the rank matrix. 445 is the feasible value generated from Gaverin Phase 1. The paths required for the solution are available in the path matrix.
ATTACHMENT D – MONTE CARLO SIMULATION DATA

The following MS Excel workbook contains the raw data used to generate the graphs for Figures Fig. 17, Fig. 18, Fig. 19, Fig. 20, Fig. 21, Fig. 32, Fig. 33, and Fig. 34.

VITA

Timothy M. Chávez

EDUCATION

Bachelor of General Studies (May 1992) in Computer Science, University of Nebraska at Omaha, Nebraska.

Master of Military Art and Science, Air War College (December 2004), Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.

PUBLICATIONS


PRESENTATIONS AT PROFESSIONAL MEETINGS


PROFESSIONAL MEMBERSHIP

Armed Forces Communications and Electronics Association – International.

Military Officers Association of America