

SMART GRID OPTIMIZATION USING A CAPACITATED TRANSSHIPMENT PROBLEM SOLVER

A Thesis  
Submitted to the Graduate Faculty  
Of the  
North Dakota State University  
Of Agriculture and Applied Science

By

Damian Lampl

In Partial Fulfillment of the Requirements  
for the Degree of  
MASTER OF SCIENCE

Major Department:  
Computer Science

July 2013

Fargo, North Dakota

North Dakota State University  
Graduate School

---

**Title**

SMART GRID OPTIMIZATION USING A CAPACITATED TRANSSHIPMENT  
PROBLEM SOLVER

---

**By**

Damian Lampl

---

The Supervisory Committee certifies that this *disquisition*  
complies with North Dakota State University's regulations and  
meets the accepted standards for the degree of

**MASTER OF SCIENCE**

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard

---

Chair

Dr. William Perrizo

---

Dr. Brian Slator

---

Dr. Samee Khan

---

Approved:

7/1/13

---

Date

Dr. Brian M. Slator

---

Department Chair

## **ABSTRACT**

A network flow model known as the capacitated transshipment problem, or CTP, can represent key aspects of a smart grid test network with the goal of finding minimum cost electric power flows using multiple different cost performance metrics.

A custom CTP Solver was developed and implemented as an ASP.NET web application in an effort to study these various minimum cost smart grid problems and provide their optimal solutions.

The CTP Solver modifies traditional linear programming concepts by introducing object oriented software development practices, as well as an insightful innovation for handling bidirectional arcs, which effectively halves the required disk and memory allocation of fully bidirectional networks.

As an initial step toward smart grid optimization problem solutions, the CTP Solver provides a glimpse of how self-healing and possibly other key components of smart grid architecture might be handled in the future.

# TABLE OF CONTENTS

ABSTRACT.....	iii
LIST OF TABLES.....	xi
LIST OF FIGURES.....	xii
LIST OF APPENDIX FIGURES.....	xiv
1. INTRODUCTION.....	1
1.1. Background.....	1
1.2. Research Focus.....	1
1.3. Organization.....	2
2. LITERATURE REVIEW.....	3
2.1. Smart Grid.....	3
2.1.1. Overview.....	3
2.1.2. Self-Healing System.....	4
2.1.3. IEEE Test Systems.....	5
2.2. Linear Programming.....	6
2.2.1. Overview.....	6
2.2.2. Simplex Method.....	7
2.2.2.1. General Form.....	8
2.2.2.2. Standard Form.....	8
2.2.3. Big-M Method.....	10
2.2.4. Interior-Point Method.....	11
2.2.5. Duality.....	11
2.2.6. Network Flow Problems.....	12
2.2.6.1. Network Simplex Algorithm.....	13
2.2.7. Capacitated Transshipment Problem.....	14
2.2.7.1. Transportation Problem.....	14
2.2.7.2. Transshipment Nodes.....	15
2.2.7.3. Arc Capacities.....	15

2.2.7.4. CTP Standard Form .....	15
2.2.8. Test Networks .....	16
2.2.9. CTP and the Smart Grid .....	17
3. IMPLEMENTATION .....	19
3.1. Design Goals .....	19
3.1.1. User Experience .....	20
3.1.1.1. User Conveniences .....	20
3.1.2. Application Development and Maintenance .....	21
3.1.2.1. Implementation Goals .....	21
3.2. Implementation Overview .....	22
3.2.1. Simplified Process .....	22
3.2.2. Smart Grid Possibilities .....	23
3.3. Data Structures .....	23
3.3.1. DataSet .....	24
3.3.2. IList .....	24
3.3.3. LINQ .....	25
3.3.4. Node .....	25
3.3.5. Arc .....	28
3.3.6. Algorithm Methods .....	32
3.3.7. Miscellaneous Data Structures .....	34
3.3.8. XML Input Files .....	35
3.4. Modified Simplex Algorithm .....	36
3.4.1. Step 1: Initialize .....	37
3.4.1.1. Set Root Node .....	37
3.4.1.2. Create Basis Tree .....	38
3.4.1.3. Determine Node Potentials .....	40
3.4.2. Step 2: Calculate Reduced Costs .....	40
3.4.2.1. Non-basic Arcs .....	41
3.4.2.2. Bidirectional Arcs .....	41

3.4.2.3. Choose Entering Arc .....	42
3.4.2.4. Enforce Lower Bounds .....	43
3.4.2.5. Optimality .....	43
3.4.3. Step 3: Create Cycle .....	43
3.4.3.1. Add Arcs to Cycle .....	44
3.4.3.2. Calculate Maximum Feasible Flow Change .....	45
3.4.3.3. Choose Leaving Arc .....	46
3.4.3.4. Update Cycle Flows .....	46
3.4.3.5. Degeneracy .....	47
3.4.4. Step 4: Update Basis .....	47
3.4.5. Step 5: Repeat Steps 2-4 Until Optimal .....	48
3.5. Output .....	49
3.5.1. Miscellaneous Information .....	49
3.5.2. Optimal Solution .....	50
3.5.3. Simplex Iterations .....	52
3.5.4. Debugging Log .....	55
3.6. Limitations and Modifications .....	56
3.6.1. Performance Gains .....	56
3.7. Network Generator .....	57
3.7.1. IEEE Test Files .....	57
3.7.1.1. Formatting Discrepancies .....	58
3.7.1.1.1. Duplicate Arcs .....	58
3.7.1.1.2. Sequential Node ID Requirement .....	59
3.7.1.2. Supply and Demand .....	59
3.7.2. Generic Networks .....	60
3.7.2.1. Realistic Networks .....	60
4. RESULTS .....	62
4.1. Software Comparisons .....	62
4.1.1. AMPL .....	62

4.1.2. SAS .....	63
4.1.3. CTP Solver .....	63
4.1.3.1. Modeling .....	63
4.1.3.2. Software Installation .....	64
4.1.3.3. Output .....	64
4.2. Reading the CTP Solver Results .....	64
4.2.1. Optimal Network .....	65
4.2.2. Simplex Iterations .....	66
4.2.2.1. Nodes .....	66
4.2.2.2. Current Network Values .....	67
4.2.2.3. Entering and Leaving Arcs .....	67
4.2.2.4. Basis .....	67
4.2.2.5. Cycle Arcs .....	68
4.2.2.6. Network Cost .....	68
4.2.3. Cycle Debugging Log .....	68
4.2.4. Displayed Results Comparison .....	68
4.2.4.1. Optimal Display and Diagnostics Comparison .....	69
4.2.4.2. Optimal Network Comparison .....	71
4.3. Accuracy Summary .....	75
4.4. Performance Summary .....	75
4.4.1. CTP Solver Output Performance Improvements .....	76
4.4.1.2. Hiding Simplex Iterations .....	77
4.4.1.3. Hiding Cycle Debugging Log .....	77
4.4.1.4. Hiding Cycle Debugging Log and Simplex Iterations .....	78
4.4.1.5. First Reduced Cost Arc .....	79
4.5. Testing Environment and Setup .....	79
4.5.1. Hardware and Software .....	79
4.5.2. Test Network Setup .....	80
4.6. Test Network Results .....	80

4.6.1. IEEE 14-Bus Results: Distance Cost .....	81
4.6.1.1. Accuracy .....	82
4.6.1.2. Performance .....	82
4.6.2. IEEE 30-Bus Results: Random Cost .....	83
4.6.2.1. Accuracy .....	83
4.6.2.2. Performance .....	84
4.6.2.3. Performance Graphs .....	84
4.6.3. IEEE 57-Bus Results: Random Cost .....	85
4.6.3.1. Accuracy .....	85
4.6.3.2. Performance .....	85
4.6.3.3. Performance Graphs .....	86
4.6.4. IEEE 118-Bus Results: Random Cost .....	86
4.6.4.1. Accuracy .....	87
4.6.4.2. Performance .....	87
4.6.4.3. Performance Graphs .....	88
4.6.5. IEEE 300-Bus Results: Random Cost .....	88
4.6.5.1. Accuracy .....	88
4.6.5.2. Performance .....	88
4.6.5.3. Performance Graphs .....	89
4.6.6. Custom 400 Node Results: Random Cost .....	89
4.6.6.1. Accuracy .....	90
4.6.6.2. Performance .....	90
4.6.6.3. Performance Graphs .....	91
4.6.7. Custom 500 Node Results: Random Cost .....	91
4.6.7.1. Accuracy .....	91
4.6.7.2. Performance .....	91
4.6.7.3. Performance Graphs .....	92
4.7. Results Analysis .....	92
5. CONCLUSION .....	95



5.1. Primary Contributions.....	96
5.1.1. Architecture and Platform.....	96
5.1.1.1. Web Application.....	96
5.1.1.2. Object-Oriented Architecture and Concepts.....	97
5.1.1.3. LINQ.....	97
5.1.2. User Experience.....	97
5.1.2.1. Ease of Use.....	98
5.1.2.2. Standardized Data Format.....	98
5.1.2.3. No New Languages.....	98
5.1.3. Bidirectional Arcs Algorithm Innovation.....	99
5.2. Future Work and Improvements.....	99
5.2.1. Parallelization.....	99
5.2.2. Visualization.....	100
5.2.3. Optimization.....	100
REFERENCES.....	102
APPENDIX.....	104
A.1. IEEE 14-Bus System Example.....	104
A.1.1. IEEE 14-Bus System: Step 0.....	104
A.1.2. IEEE 14-Bus System: Step 1 Cycle.....	105
A.1.3. IEEE 14-Bus System: Step 1 Flows.....	106
A.1.4. IEEE 14-Bus System: Step 2 Cycle.....	106
A.1.5. IEEE 14-Bus System: Step 2 Flows.....	107
A.1.6. IEEE 14-Bus System: Step 3 Cycle.....	107
A.1.7. IEEE 14-Bus System: Step 3 Flows.....	108
A.1.8. IEEE 14-Bus System: Step 4 Cycle.....	108
A.1.9. IEEE 14-Bus System: Step 4 Flows.....	109
A.1.10. IEEE 14-Bus System: Step 5 Cycle.....	109
A.1.11. IEEE 14-Bus System: Step 5 Flows.....	110
A.1.12. IEEE 14-Bus System: Step 6 Cycle.....	110

A.1.13.	IEEE 14-Bus System: Step 6 Flows .....	111
A.1.14.	IEEE 14-Bus System: Step 7 Cycle .....	111
A.1.15.	IEEE 14-Bus System: Step 7 Flows .....	112
A.1.16.	IEEE 14-Bus System: Step 8 Cycle .....	112
A.1.17.	IEEE 14-Bus System: Step 8 Flows .....	113
A.1.18.	IEEE 14-Bus System: Step 9 Cycle .....	113
A.1.19.	IEEE 14-Bus System: Step 9 Flows .....	114
A.1.20.	IEEE 14-Bus System: Step 10 Cycle .....	114
A.1.21.	IEEE 14-Bus System: Step 10 Flows .....	115
A.1.22.	IEEE 14-Bus System: Step 11 Cycle .....	115
A.1.23.	IEEE 14-Bus System: Step 11 Flows .....	116
A.1.24.	IEEE 14-Bus System: Step 12 Cycle .....	116
A.1.25.	IEEE 14-Bus System: Step 12 Flows .....	117
A.1.26.	IEEE 14-Bus System: Step 13 Cycle .....	117
A.1.27.	IEEE 14-Bus System: Step 13 Flows .....	118
A.1.28.	IEEE 14-Bus System: Step 14 Cycle .....	118
A.1.29.	IEEE 14-Bus System: Step 14 Flows .....	119
A.1.30.	IEEE 14-Bus System: Step 15 Cycle .....	119
A.1.31.	IEEE 14-Bus System: Step 15 Flows .....	120
A.1.32.	IEEE 14-Bus System: Step 16 Cycle .....	120
A.1.33.	IEEE 14-Bus System: Step 16 Flows (Optimal) .....	121

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
3.1: Node Properties.....	26
3.2: Node Methods.....	28
3.3: Arc Properties.....	29
3.4: Arc Methods.....	31
3.5: Algorithm Methods.....	33
4.1: CTP Solver Performance Improvement - Hiding Simplex Iterations.....	77
4.2: CTP Solver Performance Improvement - Hiding Cycle Debugging Log.....	78
4.3: CTP Solver Performance Improvement - Hiding Cycle Debugging Log and Simplex Iterations.....	78
4.4: IEEE 14- Bus Performance Results.....	82
4.5: IEEE 30-Bus Performance Results.....	84
4.6: IEEE 57-Bus Performance Results.....	86
4.7: IEEE 118-Bus Performance Results.....	87
4.8: IEEE 300-Bus Performance Results.....	88
4.9: Custom 400 Node Performance Results.....	90
4.10: Custom 500 Node Performance Results.....	92

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1: Example Smart Grid Visualization. [2].....	3
2.2: Dynamic Electrical Power Rerouting. [2].....	5
2.3: Linear Programming Model General Form. [6].....	8
2.4: Linear Programming Model Standard Form. [6].....	9
2.5: Linear Programming Model Simplified Standard Form. [6].....	10
2.6: Linear Programming Model Vector Standard Form. [6].....	10
2.7: Primal and Dual Problems. [6].....	12
2.8: CTP Standard Form.....	16
2.9: Left - IEEE 14-Bus Test System Diagram. Right - IEEE 14-Bus Test System Network Representation.....	17
3.1: CTP Solver (Screenshot, Light CSS).....	19
3.2: Initial Basis Tree of IEEE 14-Bus Test System with Arc Flows.....	38
3.3: IEEE 14-Bus Test System Cycle Iteration 13.....	44
3.4: IEEE 14-Bus Test System Basis Update 13.....	48
3.5: CTP Solver Optimal Network Miscellaneous Information (Screenshot, Light CSS).....	50
3.6: CTP Solver Optimal Network Arc Information (Screenshot, Light CSS)....	51
3.7: CTP Solver IEEE 14-Bus System Simplex Iteration 13 - Node Details (Screenshot, Light CSS).....	53
3.8: CTP Solver IEEE 14-Bus System Simplex Iteration 13 - Full Network Results (Screenshot, Light CSS).....	54
3.9: CTP Solver IEEE 14-Bus System Simplex Iteration 13 - Entering/Leaving Arcs, Basis, Cycle, and Current Network Cost (Screenshot, Light CSS)..	55
4.1: AMPL Optimal and Diagnostics Display (Screenshot).....	69
4.2: SAS Optimal and Diagnostics Display (Screenshot).....	70
4.3: CTP Solver Optimal and Diagnostics Display (Screenshot, Light CSS)....	70
4.4: AMPL Optimal Network (Screenshot).....	72
4.5: SAS Optimal Network (Screenshot).....	73

4.6: CTP Solver Optimal Network (Screenshot, Light CSS).....	74
4.7: IEEE 14-Bus Performance Graphs.....	83
4.8: IEEE 30-Bus Performance Graphs.....	84
4.9: IEEE 57-Bus Performance Graphs.....	86
4.10: IEEE 118-Bus Performance Graphs.....	88
4.11: IEEE 300-Bus Performance Graphs.....	89
4.12: Custom 400 Node Performance Graphs.....	91
4.13: Custom 500 Node Performance Graphs.....	92
4.14: Overall Average Performance.....	93

## LIST OF APPENDIX FIGURES

<u>Figure</u>	<u>Page</u>
A.1: IEEE 14-Bus System Step 0.....	105
A.2: IEEE 14-Bus System Step 1 Cycle.....	105
A.3: IEEE 14-Bus System Step 1 Flows.....	106
A.4: IEEE 14-Bus System Step 2 Cycle.....	106
A.5: IEEE 14-Bus System Step 2 Flows.....	107
A.6: IEEE 14-Bus System Step 3 Cycle.....	107
A.7: IEEE 14-Bus System Step 3 Flows.....	108
A.8: IEEE 14-Bus System Step 4 Cycle.....	108
A.9: IEEE 14-Bus System Step 4 Flows.....	109
A.10: IEEE 14-Bus System Step 5 Cycle.....	109
A.11: IEEE 14-Bus System Step 5 Flows.....	110
A.12: IEEE 14-Bus System Step 6 Cycle.....	110
A.13: IEEE 14-Bus System Step 6 Flows.....	111
A.14: IEEE 14-Bus System Step 7 Cycle.....	111
A.15: IEEE 14-Bus System Step 7 Flows.....	112
A.16: IEEE 14-Bus System Step 8 Cycle.....	112
A.17: IEEE 14-Bus System Step 8 Flows.....	113
A.18: IEEE 14-Bus System Step 9 Cycle.....	113
A.19: IEEE 14-Bus System Step 9 Flows.....	114
A.20: IEEE 14-Bus System Step 10 Cycle.....	114
A.21: IEEE 14-Bus System Step 10 Flows.....	115
A.22: IEEE 14-Bus System Step 11 Cycle.....	115
A.23: IEEE 14-Bus System Step 11 Flows.....	116
A.24: IEEE 14-Bus System Step 12 Cycle.....	116
A.25: IEEE 14-Bus System Step 12 Flows.....	117
A.26: IEEE 14-Bus System Step 13 Cycle.....	117

A.27: IEEE 14-Bus System Step 13 Flows.....	118
A.28: IEEE 14-Bus System Step 14 Cycle.....	118
A.29: IEEE 14-Bus System Step 14 Flows.....	119
A.30: IEEE 14-Bus System Step 15 Cycle.....	119
A.31: IEEE 14-Bus System Step 15 Flows.....	120
A.32: IEEE 14-Bus System Step 16 Cycle.....	120
A.33: IEEE 14-Bus System Step 16 Flows (Optimal).....	121

# 1. INTRODUCTION

## 1.1. Background

The creation of an autonomous, self-healing electrical grid is currently one of the most important challenges facing electrical energy providers. Such a system, known as the "smart grid", must interweave a multitude of different systems, both software and hardware, in order to form a complete solution capable of meeting the requirements outlined by the United States Department of Energy.

According to the DOE, "It is a colossal task. But it is a task that must be done." [1]

## 1.2. Research Focus

This work focuses on a single aspect of those systems: optimal electrical flow through the smart grid network as determined by a cost factor. The cost factor can be a different performance metric for various optimization objectives, including values such as the distance between generators and customers, electric line repair times, or failure rates.

In order to determine the best solution for multiple, different cost-related problems associated with the smart grid, the capacitated transshipment problem, or CTP, was chosen from the mathematical field of linear programming to model the smart grid network and its values. Using this model, a custom CTP Solver was developed, allowing users to easily determine the optimal network flow of a given smart grid network topology.



### **1.3. Organization**

Four chapters describe the work performed, beginning with a short literature review, followed by the custom CTP Solver's implementation, results, and conclusion.

The literature review chapter focuses on the problem definition, including an overview of the smart grid, linear programming, and how the capacitated transshipment problem is applied to the self-healing aspect of the smart grid.

The implementation chapter focuses on the steps involved with creating the CTP Solver and the reasons behind its design and implementation decisions. Each step of its modified simplex algorithm is thoroughly described as to be easily followed. In addition to the details of the CTP Solver, a description of the custom network generator used in the creation of larger-scale networks is also included.

In the results chapter, the CTP Solver's output is explained and its performance is compared to a couple of existing linear programming software solutions: AMPL and SAS. The accuracy as far as the minimum cost network attained for each test network is also compared against AMPL and SAS, but since they all implement optimal algorithms, the execution time of the algorithm is a more important comparison factor.

Finally, the conclusion chapter sums up the work and includes recommendations as well as opportunities for future work and improvements to the CTP Solver and network generator.

## 2. LITERATURE REVIEW

For a better understanding of the purpose of the CTP Solver and this work, some background information on the smart grid and linear programming will be helpful. In particular, the smart grid, the simplex algorithm, network flow problems, and especially the capacitated transshipment problem itself will be reviewed in some detail.

### 2.1. Smart Grid



Figure 2.1: Example Smart Grid Visualization. [2]

#### 2.1.1. Overview

The term "the grid," refers to the electric power grid of the United States. It is a network consisting of substations, transformers, and transmission lines used to provide electricity to homes and businesses from an electric power plant. [2]

The existing electric grid originated in the 1890s and has both grown and evolved to a network of more than 9,200 electric generating units. These generators are capable of producing over one million megawatts of generating

capacity, made available through more than 300,000 miles of transmission lines. [2]

While the current grid is considered an engineering marvel [2], it is beginning to show its age. As our electricity needs and demands increase and advance, so, too, must the electric grid providing the power.

It follows, then, that "smart grid" refers to using computer-based remote control and automation in an effort to modernize the utility electricity delivery systems. [3] Among the many benefits these automated systems would help improve is the reliability of the electrical grid by dynamically rerouting power as needed in order to avoid cascading failures.

### **2.1.2. Self-Healing System**

One of the greatest benefits of a fully-functional smart grid is the concept of self-healing. Current methods of outage detection vary and can be primitive at best, requiring customers to call the electric provider with service interruption notifications.

This type of recovery solution is completely reactive, and often times much too slow to prevent catastrophic failures such as cascading outages. When a generator fails, a large system is affected and can cause overloading of other generators. As stations continue to fail, the outage spreads farther and farther throughout the network.

Self-healing in the smart grid is just one aspect of a larger concept referred to as "distribution intelligence." It is concerned with the utility distribution system, or the wires, switches, and transformers that connect the utility substation to the customers. [2]

Outage detection is another aspect of smart grid distribution intelligence. The CTP Solver assumes an outage has been detected and concerns itself with the optimal redistribution of power based on the current state of the smart grid network.



Figure 2.2: Dynamic Electrical Power Rerouting. [2]

### **2.1.3. IEEE Test Systems**

The Institute of Electrical and Electronics Engineers is the world's largest professional association dedicated to advancing technological innovation and excellence for the benefit of humanity. [4] The IEEE has multiple test systems available for the study of electrical grid networks, generally distinguished by a differing number of busses.

The bus system files used in testing the CTP Solver include:

- 14-Bus Test System
- 30-Bus Test System
- 57-Bus Test System
- 118-Bus Test System
- 300-Bus Test System

Diagrams and network data files are available on the University of Washington Electrical Engineering website for each of the IEEE test systems above. [5]

## **2.2. Linear Programming**

### **2.2.1. Overview**

Sometimes referred to as "linear optimization", linear programming can be defined as the general approach to the modeling and solution of linear mathematical models. [6, p. 2]

The term "programming" can be a bit misleading since it does not specifically mean *computer programming*, which many people might assume at first glance. In this context, it provides a more general reference to problem solutions; of course, these solutions could in fact be implemented as computer programs but that is not a requirement.

Three basic steps are usually followed when formulating a model to represent a given linear programming problem:

1. Determination of the decision variables
2. Formulating the objective function
3. Formulating the constraints

The decision variables represent measurable aspects of the problem, such as unit cost. The objective function seeks to optimize the problem and the constraints are limitation requirements.

With a model in place, the key concept of linear programming is optimization of the objective function, which can also be thought of in terms of minimization or maximization. When feasible, the solution to a linear programming problem will be the best possible result of the objective function value with respect to any constraints.

Some canonical examples of linear programming problems include the assignment problem, the traveling salesman problem, and the transportation problem which is a simplified variation of the capacitated transshipment problem.

There are currently a number of software solutions that focus on solving linear programming and optimization problems, including AMPL and SAS.

### **2.2.2. Simplex Method**

The simplex method, sometimes referred to as the simplex algorithm, is an algebraic process for solving linear programming problems. George Bernard Dantzig is considered the creator of the simplex method, first published in 1947 and detailed in a 1951 Cowles Commission for Research in Economics conference. [7]

To summarize its concepts, the simplex method mathematically models a problem so that its solution space can be described in one of three ways:

1. Feasible Solution
2. Infeasible Solution
3. Optimal Solution

An infeasible solution is simply any point that does not satisfy every constraint and nonnegativity condition of the linear program.

A feasible solution is any point that satisfies every constraint and nonnegativity condition of the linear program. The set of all feasible solutions is known as the feasible region; this is the equivalent of the intersection of all feasible solutions.

If the linear program has a bounded feasible region, meaning the feasible solution space is fully contained, an optimal solution will be some point on the feasible region boundary. The simplex method effectively traverses the boundary in search of these optimal points, also referred to as extreme points.

### 2.2.2.1. General Form

Once the basic modeling steps are complete and the decision variables, objective function, and constraints have been determined, the model can be represented mathematically.

Simply stated, the general form of a linear programming model seeks to find values for the decision variables that will optimize the objective function, subject to the problem's constraints.

Mathematically, this can be written as Figure 2.3.

<p>Optimize <math>z = c_1x_1 + c_2x_2 + \dots + c_nx_n</math></p> <p>Subject To:</p> $\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n &\{\leq, =, \geq\} b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n &\{\leq, =, \geq\} b_2 \\ &\vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n &\{\leq, =, \geq\} b_m \\ &x_1, x_2, \dots, x_n \geq 0 \end{aligned}$ <p>Where:</p> <p><math>z</math> = the objective function value <math>c</math> = cost of decision variable <math>n</math> <math>x</math> = decision variable <math>n</math> <math>a</math> = constraint <math>m, n</math> on decision variable <math>x_n</math> <math>b</math> = sum total value of constraint <math>a_{m,n}</math></p>
--

Figure 2.3: Linear Programming Model General Form. [6]

To summarize, the objective function minimizes or maximizes the sum total cost of all decision variables, subject to restrictions defined in terms of the decision variables. All decision variables must have a value greater than or equal to zero.

### 2.2.2.2. Standard Form

Since it is generally easier to solve equations instead of inequalities, additional variables, commonly referred to as artificial

variables, can be introduced into the general form of a linear programming model, producing the standard form.

These new variables are known as the surplus and slack variables. If a constraint definition is in the form of a "less-than or equal-to" inequality, a slack variable is added in order to balance its left- and right-hand sides. If a constraint is "greater-than or equal-to", a surplus variable is subtracted.

In other words, a slack variable represents the amount the left side of a constraint is missing in order to make it a balanced equation, and a surplus variable represents the amount the left side of a constraint has in excess over the right side.

For simplicity, the objective function of a linear programming model in standard form is considered a maximization problem. The slack and surplus variables can be represented in the objective function as follows in Figure 2.4.

$$\text{Maximize } z = \sum c_j x_j + \sum c_k s_k$$

Where:

$z$  = the objective function value

$c$  = cost of decision variable  $j$  or surplus/slack variable  $k$

$x$  = decision variable  $j$

$s$  = slack or surplus variable  $k$

Figure 2.4: Linear Programming Model Standard Form. [6]

In many cases, the sum of the slack and surplus variables is considered to be zero and are subsequently omitted from the notation, leaving the following Figure 2.5 as the standard form of a linear program.



Maximize  $z = \sum c_j x_j$

Subject To:

$\sum a_{ij} x_j = b_i; i = 1, \dots, m$   
 $x_j \geq 0; j = 1, \dots, n$

Where:

$z$  = the objective function value  
 $c$  = cost of decision variable  $j$   
 $x$  = decision variable  $j$   
 $a$  = constraint  $i, j$  on decision variable  $j$   
 $b$  = sum total value of constraint  $a_{ij}$

Figure 2.5: Linear Programming Model Simplified Standard Form. [6]

The standard form can also be represented using vector and matrix notation as follows in Figure 2.6.

Maximize  $z = \mathbf{c}\mathbf{x}$

Subject To:

$\mathbf{A}\mathbf{x} = \mathbf{b}$   
 $\mathbf{x} \geq \mathbf{0}$

Where:

$\mathbf{A}$  =  $m \times n$  matrix of the coefficients of the constraints  
 $\mathbf{x}$  =  $n$ -vector of decision variables  
 $\mathbf{b}$  =  $m$ -vector of constraint totals  
 $\mathbf{c}$  =  $n$ -vector of decision variable coefficients

Figure 2.6: Linear Programming Model Vector Standard Form. [6]

Formulating the model in terms of vectors and matrices is beneficial when building computer programs due to the generic array structure of most high level programming languages. A one-dimensional array is analogous to a vector, and a two-dimensional array is analogous to a matrix.

### 2.2.3. Big-M Method

Once the linear programming model has been determined, the simplex algorithm can be used to iterate through the feasible solutions until it

finds the optimal solution. But the simplex algorithm must have a starting point before it can carry out its iterations.

One way of calculating an initial solution is known as the Big-M method. In short, the Big-M method introduces artificial variables with extremely high cost coefficients, essentially guaranteeing they will not be a part of the final solution so it will only consist of real variables.

As the simplex method iterates through the feasible solutions, the artificial variables are systematically pushed out of the problem since the real variables will provide a lower-cost solution.

#### **2.2.4. Interior-Point Method**

An alternative approach to the simplex method is the interior-point method, developed in 1984 by Narendra Karmarkar. [8] Instead of following the boundary of the feasible region like the simplex method, the interior-point method constructs a trajectory through the feasible region in order to find the optimal solution.

The interior-point method is used by the "netflow" procedure in SAS whereas the "LPSOLVE" AMPL solver is based on the modified simplex algorithm and the CTP Solver uses a custom version of the simplex method.

This work is not a comparison between the simplex method and interior-point method, but rather a comparison between the CTP Solver and other optimization software to determine if it is a viable solution for smart grid optimization.

#### **2.2.5. Duality**

An important concept in linear programming is the correlation between a problem in standard form, referred to as the primal, and a related problem known as the dual, as shown in Figure 2.7.

Primal Problem	Dual Problem
Maximize $z = \mathbf{c}\mathbf{x}$ Subject To: $\mathbf{A}\mathbf{x} = \mathbf{b}$ $\mathbf{x} \geq \mathbf{0}$	Minimize $Z = \mathbf{\pi}\mathbf{b}$ Subject To: $\mathbf{\pi}\mathbf{A} \geq \mathbf{c}$ $\mathbf{\pi} \geq \mathbf{0}$
Where: $\mathbf{A}$ = $m \times n$ matrix of the coefficients of the constraints $\mathbf{x}$ = $n$ -vector of decision variables $\mathbf{b}$ = $m$ -vector of constraint totals $\mathbf{c}$ = $n$ -vector of decision variable coefficients $\mathbf{\pi}$ = $m$ -vector of dual variables	

Figure 2.7: Primal and Dual Problems. [6]

Notice that the primal in standard form is a maximization problem of the objective function while the dual is a minimization problem. The vector of decision variables in the dual is the same as the vector of right-hand sides (constraint totals, or  $\mathbf{b}$ ) in the primal, the vector of constraint totals  $\mathbf{c}$  in the dual is the same as the vector of decision variable coefficients in the primal, and the constraint coefficient matrix  $\mathbf{A}$  in the dual is the transpose of  $\mathbf{A}$  in the primal.

From these relationships, it is evident that if the primal problem is of the order  $m \times n$ , the dual problem is of the order  $n \times m$ . Likewise, when the solution to the primal problem is known, the solution to the dual problem is also known, and vice versa.

Therefore, if a primal problem has more decision variables than constraints, it might be faster to solve the dual problem instead of the primal since the solution to one provides the solution to the other.

#### 2.2.6. Network Flow Problems

In general, a network flow problem is any from a particular class in which the solution space can be described using nodes and arcs connecting

those nodes with unit flow along the arcs transferred from one node to another.

"One of the keys to developing an efficient algorithm for this class of linear programming problems is establishing a relationship between the algebraic and graphical representations of basic solutions. In particular, one of the most important relationships is the one that exists between basis matrices and rooted spanning trees." [6, p. 320]

Theorem 9.1: Every rooted spanning tree is a basis [6, p. 320]

Theorem 9.3: Every basis is a rooted spanning tree [6, p. 322]

A basis is defined as a collection of vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  in an  $n$ -dimensional (real) Euclidean space, denoted by  $R^n$ , where the following conditions hold:

1.  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  span  $R^n$ .
2. If any of these vectors is deleted, the remaining collections of vectors does not span  $R^n$ . [9, pp. 48-49]

Recall the standard form of a linear programming problem can be written in the same vector notation and an initial basis tree can be calculated using the Big-M method, allowing the simplex algorithm to be applied to network flow problems.

#### 2.2.6.1. Network Simplex Algorithm

Once the basis tree of a network flow problem has been established, the network simplex algorithm can be implemented. Network flow problems are typically considered minimization problems, although they can be easily changed into maximization problems by using negative cost values and changing the sign polarity to positive values once the solution is obtained.

The four basic steps of the network simplex algorithm are:

1. Determine the primal and dual solutions.
2. Check for optimality.

3. Determine the departing variable.
4. Pivot and update.

Modified versions of these steps will be detailed in the Implementation section.

### **2.2.7. Capacitated Transshipment Problem**

The capacitated transshipment problem, or CTP, is an important network optimization problem [9, p. 513] consisting of four primary elements: supply nodes, demand nodes, transshipment nodes, and connective arcs.

The basic concept of the CTP is to find a minimum cost path that connects every node of the network and transfers all units of flow from the supply nodes to the demand nodes without violating any network arc capacities.

#### **2.2.7.1. Transportation Problem**

The capacitated transshipment problem can be easily understood through a simplified network flow variation known as the transportation problem [6, p. 350]. In the transportation problem, unit flow is pushed along the network arcs from the supply nodes to the demand nodes.

All units of supply in the network must be transferred from the supply nodes to the demand nodes. This is known as the flow balance constraint and can be written as Equation 2.1.

$$\text{Flow Out} - \text{Flow In} - \text{Supply} = 0$$

Equation 2.1: Flow Balance Constraint.

Due to this constraint, if the total supply of a transportation problem is not equal to its total demand, the problem is infeasible.

The goal of the transportation problem is to find the basis tree that minimizes the total cost of the unit flow along the network.

#### 2.2.7.2. Transshipment Nodes

The capacitated transshipment problem generalizes the transportation problem by adding transshipment nodes; or nodes with zero supply or demand. This means the nodes must still be connected in the basis tree, but all flow units simply pass through the node.

Transshipment nodes can cause degenerate arcs, or arcs with a unit flow of zero. These degenerate arcs have the potential to create infinite loops and so must be handled properly in order to prevent cycling through the same solutions.

#### 2.2.7.3. Arc Capacities

In addition to transshipment nodes, the capacitated transshipment problem also adds capacities and lower bound requirements to arcs. Flow along any given arc must be at least as much as the lower bound and not more than its capacity. Violating either of these constraints will cause an infeasible solution.

Arcs with flow equal to their capacity can be considered part of the solution without taking up space in the basis tree. These arcs are referred to as non-basic arcs with bounded flow.

#### 2.2.7.4. CTP Standard Form

The capacitated transshipment problem can be described in algebraic standard form as shown in Figure 2.8.

Minimize  $z = \sum c_{ij}x_{ij}$

Subject To:

$x_{ji} - x_{ij} + b_i = 0$       for all arcs  $i,j$   
 $x_{ij} \geq 0$                       for all arcs  $i,j$   
 $x_{ij} \leq u_{ij}$                     for all arcs  $i,j$   
 $x_{ij} \geq l_{ij}$                     for all arcs  $i,j$

Where:

$c$  = arc cost  
 $x$  = arc flow  
 $u$  = arc upper bound  
 $l$  = arc lower bound  
 $b$  = node supply (negative value for demand)

Figure 2.8: CTP Standard Form.

The objective function is to minimize the sum total of all arc unit flows multiplied by their costs.

Constraint (1) ensures flow balance at every node by making sure total flow out of a node is the same as the total flow in with respect to the node's supply and demand requirements. This constraint also ensures that supply units are distributed from all supply nodes to all demand nodes, creating a zero net unit flow for the entire network.

Constraint (2) ensures all arcs have a non-negative unit flow.

Constraint (3) ensures no arc capacities or upper bound limits are violated.

Constraint (4) ensures no arc lower bound requirements are violated.

Formulating the problem in standard form allows the application of the network simplex algorithm for calculating the optimal solution.

### **2.2.8. Test Networks**

A handful of small example networks were used to test the functionality and accuracy of the CTP Solver's algorithm, with the most influential problem coming from the work of Bradley, Brown, and Graves [10].

This problem in particular provided an example of non-basic arcs with flow as well as non-zero lower bounds, helping to ensure the CTP Solver calculates the correct results through a variety of different potentially troublesome network characteristics.

### 2.2.9. CTP and the Smart Grid

Representing the smart grid using a capacitated transshipment problem model allows multiple different cost and network flow related problems to be easily solved. One of these problems in particular is the self-healing aspect of the smart grid.

When a critical failure is detected in the system, the CTP can be used to find an optimal and inherently feasible redirected path for redistributing energy throughout the smart grid. By finding the best alternative distribution path, customer outages can be minimized and rectified almost as quickly as they occur, when possible.

Figure 2.9 shows the diagram of the IEEE 14-Bus System and its network representation for use in the CTP Solver.

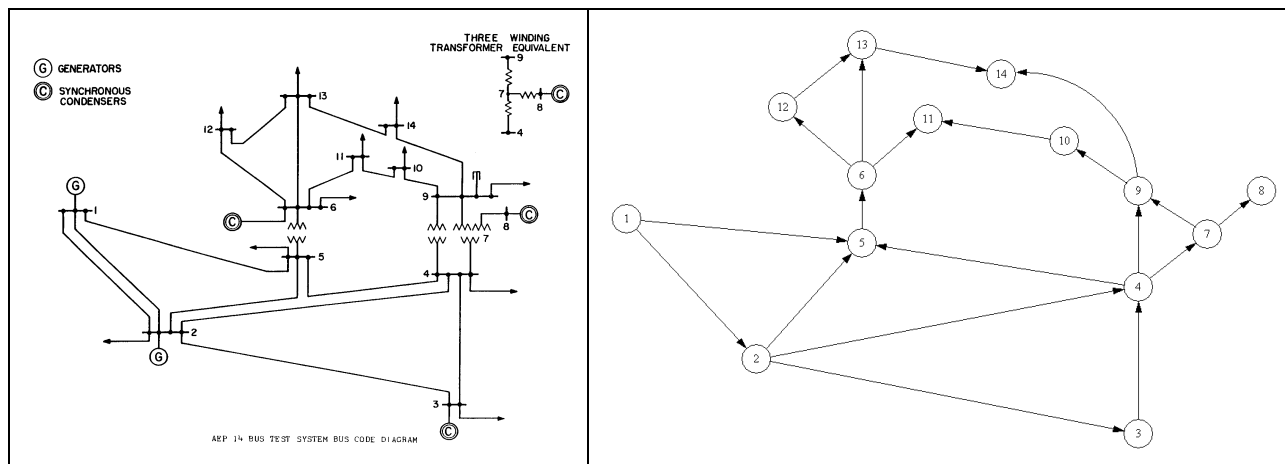


Figure 2.9: Left - IEEE 14-Bus Test System Diagram. Right - IEEE 14-Bus Test System Network Representation.



The Appendix steps through the flow updates as a high overview example of the CTP Solver's process, using the IEEE 14-Bus System with distance between the nodes (generators and substations) as a cost measurement for each arc (lines connecting generators and substations). Following the iterations can be beneficial to understanding the general capacitated transshipment problem solving process.

### 3. IMPLEMENTATION

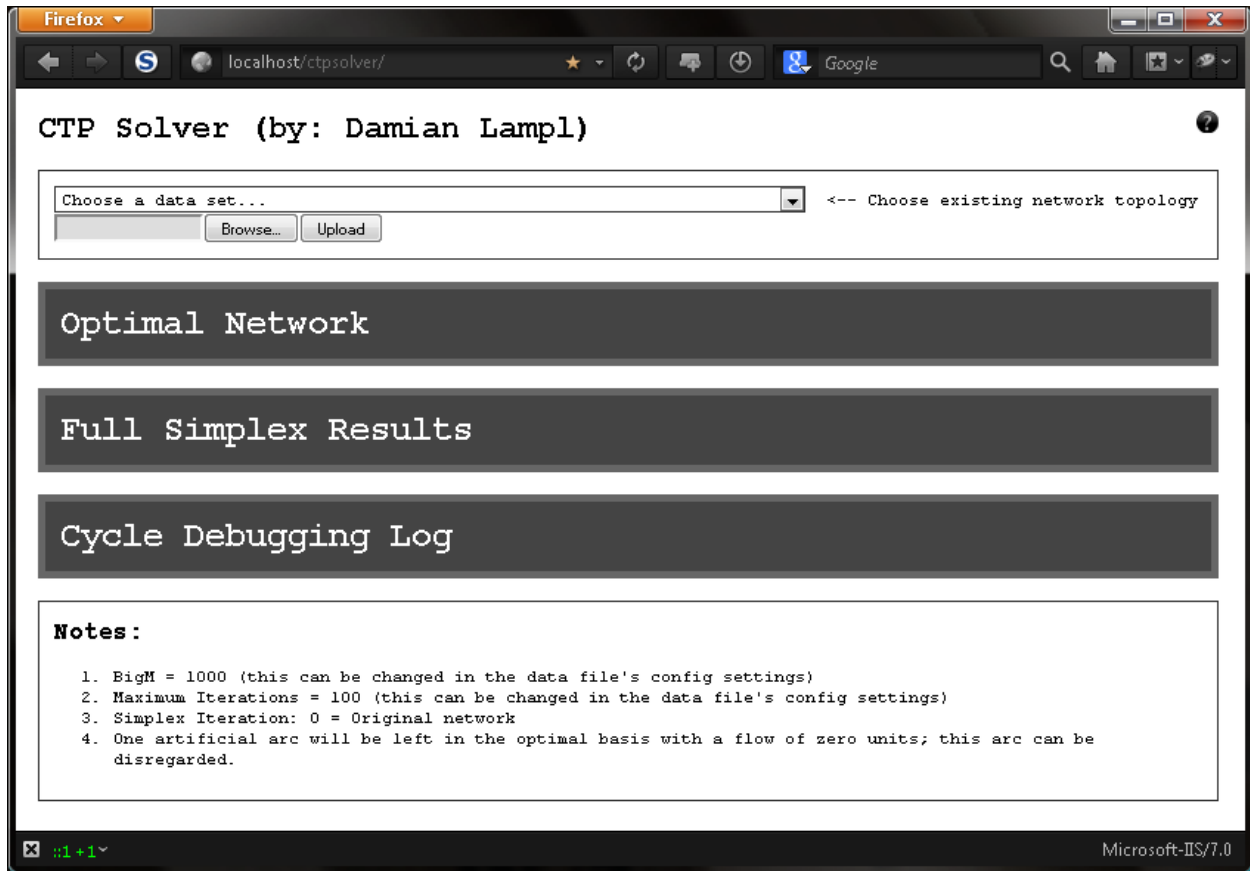


Figure 3.1: CTP Solver (Screenshot, Light CSS).

#### 3.1. Design Goals

Before getting into the nuts and bolts of the CTP Solver application, it is important to understand the motivation behind its design and architectural decisions. Once these design goals have been examined, it will hopefully be clear as to why certain development choices were implemented.

The design goals of the CTP Solver can be broken down into two primary categories:

1. User Experience
2. Application Development and Maintenance

### 3.1.1. User Experience

One of the primary factors driving the development decisions was the desire to make the application as convenient and easy to use as possible for its end users. Considerations were made for understanding how the application would generally be used and what information would be beneficial to make available for display as well as download.

The user experience design goals can be summarized in a handful of basic conveniences implemented with the user in mind.

#### 3.1.1.1. User Conveniences

- Automated Initial Basis Calculation
  - User not burdened with calculating an initial basis to feed into the network topology.
- Bi-directional Arc Capability
  - Flag allowing flow to travel in either direction along a single arc.
- Multiple Network Topology Data Formats
  - HTML Table (display)
  - Database (import)
  - XML (import, export)
  - CSV (export)
- Decimal Values
  - Enter the information as it exists instead of requiring pre-calculation transformations into integer data types, etc.
- Configuration Options
  - Big M Value
  - Simplex Iterations Limit

- Show Detailed Simplex Information
- Show Cycle Debugging Log
- Help System
- Accessibility
  - Web Application

### **3.1.2. Application Development and Maintenance**

In addition to the user experience design goals, careful considerations were made regarding the development and on-going maintenance of the CTP Solver. The purpose behind the application development and maintenance goals was to design the application's architecture and code with the developer in mind.

The application development and maintenance goals can be summarized by some basic developer-focused concepts.

#### **3.1.2.1. Implementation Goals**

- Ease of Use
  - Simplicity
  - Self-Documenting Code
  - Partial Classes
- Universal Applications
  - Generic Network Concepts
- Calculation Precision
  - Decimal Data Type
- Maintainability
  - Object Oriented Concepts

## **3.2. Implementation Overview**

The CTP Solver has been implemented as a C# ASP.NET web application, primarily for accessibility among its users. Many applications are written as standalone software and are platform dependent, or, in the case of an environment such as Java, dependent on some other service that must be installed on the end user's machine.

While dependent on the ASP.NET framework on the host server, as a web application, the CTP Solver can be made available to any machine or mobile device connected to the internet, regardless of operating system. This inherently widens the potential user-base and also minimizes the required technical capabilities of those who would most benefit from its usage.

At the same level of implementation importance as the usage ubiquity provided by the web is the ability for the CTP Solver to be as generic as possible. While it is a vital requirement to be able to handle smart grid network problems, it is equally imperative that the CTP Solver can process any capacitated transshipment problem network. This requirement has been met through basic supply, demand, flow, and cost concepts, among others.

### **3.2.1. Simplified Process**

When a user uploads a custom file or selects a network option from the dropdown of available choices, the CTP Solver reads the XML file (or database) and commits the topology to memory. The network can be fully described through two primary IList data structures: nodes and arcs, each described in more detail below.

Once the network is in memory, the CTP Solver iterates through its modified simplex algorithm and displays the final resulting optimal solution to the user. In addition to some basic computational data such as the execution time of the entire process, the optimal network is also made

available for download in both .CSV and XML format. This allows the user to easily import the optimal network into other applications.

### **3.2.2. Smart Grid Possibilities**

A benefit of web applications in specific relation to the smart grid is that an implementation could be set up in such a way that a single CTP Solver application could calculate optimal solutions for multiple client network configurations simultaneously. This capability could be of some benefit for planning systems with budget constraints as well as provide a single source of maintenance for IT staff.

Since the CTP Solver is able to connect to a database as well as read XML files, it could be easily integrated with other smart grid systems such as failure notification solutions, providing automatic optimal electric flow rerouting based on the supplied network topology of available nodes and arcs. Since arc capacities are taken into consideration, the cascading failure dynamic could possibly be avoided by ensuring network flow is feasibly rerouted.

As the smart grid system grows, more cost performance measures are likely to be revealed. Since the CTP Solver was built with generic concepts in mind, it should be able to calculate optimal results for any new network topology able to be modeled by the capacitated transshipment problem.

These potential smart grid applications and more are made possible by the CTP Solver's integration of various custom and framework-native data structures.

### **3.3. Data Structures**

Traditional linear programming techniques implement primitive data structures for network topology descriptions, such as arrays, or the standard model formulation for use in specific modeling software such as AMPL.

While these methods make sense for application speed and simplicity, the CTP Solver introduces object oriented practices in order to take advantage of robust modern programming language capabilities such as LINQ for the ASP.NET framework while maintaining very comparable speed on current hardware.

Building the application using object oriented concepts subjectively allows for easier maintainability since the primary components of the program are modularized and abstracted. Objects also make network topologies much easier for developers to conceptualize and reference during the debugging process than, for example, trying to follow array pointers in order to determine a cycle of arcs.

### **3.3.1. DataSet**

The DataSet is a native structure to the ASP.NET framework. The benefit of a DataSet object is that it takes on some properties of a traditional relational database, including concepts such as rows and columns.

The CTP Solver reads the user-supplied network into a DataSet object. In doing so, the exact same architecture can be used for reading from either XML or a traditional database, such as MSSQL or MySQL, eliminating code redundancy for essentially the same process.

### **3.3.2. IList**

An IList is another native ASP.NET data structure representing a collection of objects, with the "I" referring to the term, "Iterative", making it an iterative list. Objects stored in an IList can be "queried" much like relational databases using syntax similar to standard SQL.

Another benefit of using an IList collection object is that it only uses as much memory as it needs. Objects can be added and removed from the

collection without requiring explicit dimensions. This makes resizing the structures much more efficient than resizing arrays.

The primary limit of concern with the IList structure is the maximum number of elements allowed in a single IList object, which is the same limit as an array. Theoretically the maximum number of elements in an IList is 2,147,483,646, however, a network of that size would most likely benefit from some kind of partitioning, such as a modified Dantzig-Wolfe decomposition approach. [11]

Using a small number of these structures, the CTP Solver is able to intuitively represent the entire network topology and more.

### **3.3.3. LINQ**

The acronym, "LINQ", stands for Language-Integrated Query [12]. First introduced in Visual Studio 2008, LINQ allows strongly typed object collections (such as the IList described above) to be queried, providing an easy system for extracting relevant information from data.

LINQ queries are used generously in the custom Arc class to provide partial lists for structures such as the basis tree and all non-basic arcs.

### **3.3.4. Node**

A node is a custom object class created to represent each node of the network. All network nodes are added to an IList for easy access in calculations.

The properties of each node are described in Table 3.1.



Table 3.1: Node Properties.

Property	Data Type	Description
id	integer	<p>Providing each node with an unique id allows easy reference to individual objects as well as makes database interactions nearly seamless.</p> <p>The upper bound limit on IList collection objects is the maximum signed integer value, therefore it was logical to limit this property to an integer data type instead of a larger data type, such as long.</p> <p>The theoretical maximum number of real nodes in a network that can be calculated by the CTP Solver is 2,147,483,646 (one less than the maximum signed integer value due to the presence of a single artificial node).</p>
ConnectedArcs	IList<Arc>	List of all arcs where the node is either the Head or Tail. This list is used for calculating net flow entering and leaving the node in order to enforce flow balance requirements.
Demand	decimal	<p>The demand of a node represents the number of units of flow required at that node.</p> <p>This property is currently unused in the CTP Solver, although it is included in the node structure primarily to show it was not overlooked or mistakenly omitted.</p>
Depth	integer	<p>The depth of a node represents its level in the basis tree starting from the root node.</p> <p>It can also be explained as the number of arcs between the current node and the root node in the basis tree.</p> <p>The node depth is used in traversing the cycle created by a non-basic arc entering the basis.</p>
Name	string	<p>The name of the node is included mainly for the benefit of human readability if the network topology is printed to screen or if future functionality includes generating network diagrams, etc.</p> <p>The CTP Solver does not use this property for any purpose in its algorithms other than simply storing the information.</p>

Table 3.1: Node Properties (continued).

Property	Data Type	Description
NetFlow	decimal	<p>Defined as the total flow coming into a node, minus the total flow leaving a node, plus the node's supply (which will be subtracted when representing demand since it is then a negative value).</p> <p>This property must return zero when calculated for every node in the network, otherwise a net flow violation has occurred and the network is infeasible.</p>
Parent	integer	<p>The parent of a node is used in the basis tree structure. It represents the node immediately connected to and one depth level above the current node.</p> <p>A negative parent value represents a reflected arc in the basis tree structure.</p> <p>Since it refers to a node id property, it is also an integer data type.</p>
Potential	decimal	<p>The potential of a node is the equivalent to a dual variable in linear programming.</p> <p>In more simple terms, the node potential is the cost of the back path in the basis tree from the current node to the root node.</p> <p>This value is used in calculating the reduced cost of non-basic arcs to find the best candidate arc for entering the basis tree.</p>
Successor	integer	<p>The successor of a node is also used in the basis tree structure. It represents the node following the current node in the preorder thread. Unlike parent nodes, successor nodes in the preorder thread are not necessarily directly connected by an arc.</p>

Table 3.1: Node Properties (continued).

Property	Data Type	Description
Supply	decimal	<p>The supply of a node represents the number of units of flow available from that node.</p> <p>For simplicity in the CTP Solver algorithm, the supply property also represents the demand of a given node by reversing its polarity to a negative sign.</p> <p>Transshipment nodes are given a supply value of zero.</p>

The Node class has a single method, shown in Table 3.2.

Table 3.2: Node Methods.

Method	Description
GetArcs	Returns an IEnumerable list of arcs where the node is either the Head or Tail. This method is used to set the value of the ConnectedArcs property.

### **3.3.5. Arc**

An arc is the second custom object class used to represent connections, also sometimes referred to as edges or links, between nodes. As with nodes, all network arcs are added to an IList data structure, allowing a simple representation of the network connections.

The properties of arc objects are described in Table 3.3.

Table 3.3: Arc Properties.

Property	Data Type	Description
id	integer	<p>Providing each arc with an unique id allows easy reference to individual objects as well as makes database interactions nearly seamless.</p> <p>The upper bound limit on IList collection objects is the maximum signed integer value, therefore it was logical to limit this property to an integer data type instead of a larger data type, such as long.</p> <p>The theoretical maximum number of real arcs in a network that can be calculated by the CTP Solver is <math>(2,147,483,647 - n)</math>, where <math>n</math> is the number of nodes in the network. The reason the number of nodes are taken into account is due to the creation of an artificial arc connecting every real node to the artificial node during the automated initial basis calculation.</p>
BasisOrder	integer	<p>Originally used before the initial basis was automatically calculated, the basis order represents the arc's order in the basis tree.</p> <p>It is still used in the basis iterations and calculations; non-basic arcs are determined by a basis order value of zero.</p>
Capacity	decimal	<p>The capacity is the upper bound limit on units of flow that can move across the arc at a given time.</p> <p>The capacity adheres to the Big M maximum value limit set by the user in the config element.</p>
Cost	decimal	<p>The cost represents the price of moving one unit of flow across the arc. It is important to note that the total cost of an arc is calculated by multiplying the arc cost and flow together.</p> <p>The cost adheres to the Big M maximum value limit set by the user in the config element.</p>
Cpx	decimal	<p>Cpx is the calculated value of the capacity minus the flow of a given arc.</p>

Table 3.3: Arc Properties (continued).

Property	Data Type	Description
Flow	decimal	The flow is the number of units pushed across the arc from the tail node to the head node. Its value must fall between the capacity and lower bound of the arc.
Head	integer	The head of an arc is the id of its destination node. It is the stopping point of a directed arc. Unit flow along the arc starts at the tail node and moves toward the head node.
IsArtificial	Boolean	<p>The IsArtificial flag determines whether or not the arc is artificial or real. Artificial arcs are used in the creation of the initial basis and connect real nodes to the single artificial node.</p> <p>If an arc is artificial, it is not allowed to reenter the basis tree once it has been removed.</p>
IsBasic	Boolean	The IsBasic flag determines whether or not the arc is in the basis tree. It could be considered somewhat redundant due to the BasisOrder property, but it is used in some logic checks and output displays.
IsBidirectional	Boolean	<p>The IsBidirectional flag determines whether or not the arc can be considered to have flow move in either direction: from the tail node to the head or from the head node to the tail. Including this flag allows the network size to be effectively doubled without the need for duplicating the entire network.</p> <p>The one caveat is that a bidirectional arc must have the same properties regardless of the flow direction; so, for example, a bidirectional arc could not have a separate cost for flow moving from the head node to the tail as it does moving from the tail node to the head.</p>

Table 3.3: Arc Properties (continued).

Property	Data Type	Description
LowerBound	decimal	<p>The lower bound is the minimum units of flow required on the arc.</p> <p>The CTP Solver is able to enforce lower bounds when possible, considering those arcs first and forcing them into the network with as much flow allowed by theta, or the maximum cycle flow change.</p> <p>If an optimal solution is reached with lower bound violations, a warning message is displayed to the user.</p>
ReducedCost	decimal	<p>The reduced cost is a value for determining which non-basic arc will enter the basis. The arc with the best reduced cost, meaning the arc that will lower the overall cost of the basis by the largest amount, is chosen to enter the basis.</p>
SameCycleDirection	Boolean	<p>The same cycle direction flag determines whether or not the arc follows the same flow direction as the cycle's entering arc. It is used in calculating the maximum flow allowed along the cycle created by adding the entering arc to the basis.</p>
Tail	integer	<p>The tail of an arc is the id of its source node. It is the starting point of a directed arc. Unit flow along the arc starts at the tail node and moves toward the head node.</p>

In addition to the properties just described, the arc class has a few important methods used in the CTP Solver's algorithms, described in Table 3.4.

Table 3.4: Arc Methods.

Method	Description
GetArc	<p>Uses a LINQ query to return an arc from the provided head and tail node id's.</p>

Table 3.4: Arc Methods (continued).

Method	Description
GetBasis	Uses a LINQ query to find all basic arcs in the provided IList object and represents the basis tree structure. It returns an IOrderedEnumerable list of arcs used in procedures requiring only the basis tree.
GetChildArcs	Uses one of two LINQ queries to return the immediate basis tree arcs connected to the provided parent node.  Does not select the arc connecting the parent node and the provided grandparent node.
GetNonBasic	Uses a LINQ query to find all non-basic arcs in the provided IList object. It returns an IOrderedEnumerable list of arcs used in calculations requiring only non-basic arcs, such as determining the reduced cost.  Excludes artificial arcs.
GetNonBasicWithFlow	Uses a LINQ query to find all non-basic arcs with non-zero flow (such as upper- or lower-bounded arcs).  Only used for output display purposes.
GetReversePreorder	Uses a LINQ query to find the basis tree in reverse order. While its functionality could have been created by using a parameterized version of the GetBasis() method, a separate method helped make the purpose more clear in the calling procedures.
ResetCycleDirections	Sets the value of all arc same cycle direction flags to true for new cycle calculations.
ResetReducedCosts	Sets the value of all arc reduced cost values to zero for a new reduced cost calculation iteration.

### **3.3.6. Algorithm Methods**

The modified simplex algorithm implemented (described below) is broken down into a handful of methods as shown in Table 3.5.

Table 3.5: Algorithm Methods.

Method	Description
GetData	Container method for reading the database or XML file and initializing the list structures and other variables.
GetRecordCounts	Counts the number of nodes and arcs existing in the network.
GetConfigSettings	Reads the configuration options set by the user, including the Big M value, maximum simplex iterations, and whether or not to display the detailed information for each simplex iteration or the cycle iteration debugging.
DisplayOptimal	Builds the optimal solution table and prints the result to the screen.
DisplayData	Builds the tables displayed for each simplex iteration, including separate tables for node and arc data, entering and leaving arcs, the basis tree, and cycle arcs. The current total network cost is also included.
BuildArcTableHeader	Allows dynamic output table header creation.
GetNodes	Reads the node information from the data and builds the list of nodes as well as the initial basis.
GetArcs	Reads the arc information from the data and builds the list of arcs.
CalculateSimplex	The main calculation loop of the application. Calls helper methods for calculating the reduced cost, creating a cycle, and updating the basis for each simplex iteration.
CalculateReducedCost	Loops through non-basic arcs to find the entering arc for creating a cycle.
CreateCycle	Builds the cycle created by adding the entering arc to the basis, determines the maximum flow change, chooses and removes the leaving arc from the basis.
TraverseBackpath	Determines the basis tree arc of a given node and depth; helper method used in cycle creation.



Table 3.5: Algorithm Methods (continued).

Method	Description
UpdateBasis	Recursively updates the node preorder values, finds immediate child nodes at a given basis depth, determines arc reflection, and node depths.

### **3.3.7. Miscellaneous Data Structures**

In addition to the primary data structures, a small number of helper and utility classes were created to handle various aspects of the application.

Three utility classes were used for handling commonly used functions, user-configurable options, and database interactions.

In addition, all custom class structures were created as partial classes. Using a partial class allows its methods to be defined in separate files. It is a common practice to define an entire class in a single file using the class name as the file name. The benefit of using a partial class comes from the ability to logically separate categorized methods of a class for easier maintainability as well as allowing common generic methods to be automatically generated.

To speed the development of the creation of the node and arc database interaction methods, a custom object relationship mapping application (commonly referred to as O/R mapping or ORM) was used to read the node and arc class structures from a database and automatically generate the code for their interactions. Some common methods include getting an object or list of objects from the database, as well as inserting, modifying, and deleting object records.

Since the node and arc classes were created as partial classes, the generated database methods could be easily re-generated and stored in separate files if changes were made to the class properties. This eliminated

the need for rewriting any methods or copying and pasting code from the generated files into a single class file.

### **3.3.8. XML Input Files**

The information required by the CTP Solver to represent the network is relatively minimal. A user need only supply the following properties for the nodes:

1. id
2. Name
3. Supply (negative value used for demand)

Again, the name property is simply included for human readability and could in fact be omitted from the XML document and require a single line to be commented out in the CTP Solver's code. If memory limitations were to arise, this would be a good first step in minimizing some overhead.

The required arc properties include:

1. Tail
2. Head
3. Capacity
4. LowerBound
5. Cost
6. BiDirectional

In addition to the node and arc information, the CTP Solver also requires a configuration element. This element allows the user to set specific values for Big M and the maximum simplex iterations. Enabling these two values to be defined by the users provides some customization to the CTP Solver's capabilities without compromising the application's algorithms with problems such as infinite loops.

In addition to calculation options, the configuration element also allows the user to choose whether or not to display full output details for

each simplex iteration and/or cycle iteration debugging information. As shown in the results, disabling this optional information can provide dramatic performance gains on larger networks.

### **3.4. Modified Simplex Algorithm**

Recall the typical steps in the network simplex algorithm [9, pp. 347-348] for the capacitated transshipment problem are:

1. Determine Primal and Dual Solutions
2. Check Optimality
3. Add Lower Bounded Arc to Basis
4. Add Upper Bounded Arc to Basis

Since the CTP Solver was written with object oriented concepts in mind, it deviates from the standard linear programming model and uses a modified simplex algorithm to find the optimal solution of a given network, described in the following five steps:

1. Initialize
2. Calculate Reduced Costs
3. Create Cycle
4. Update Basis
5. Repeat Step 2 - Step 4 Until Optimal

In comparison, Step 1 and Step 2 are by and large performing the same functionality in both the typical algorithm and the CTP Solver's algorithm.

The typical Step 3 and Step 4 are essentially modified versions of the same step, making slight alterations between the way lower bounded and upper bounded arcs are handled and entered through a conditional check determined in Step 2. The CTP Solver effectively combines these two steps into its Step 3 for creating the cycle when either an upper bounded or lower bounded arc is added to the basis.

The typical Step 3 and Step 4 also break down into multiple sub-steps that include updating the basis tree. This particular process seemed to make sense as a separate subroutine and can be more easily understood as a separate step in the algorithm.

Finally, the CTP Solver's Step 5 was included as a separate algorithmic step for similar reasons as its Step 4, allowing a more simplified description of the process. As with the basis updating process, this step is also embedded as part of the typical algorithm's Step 3 and Step 4.

### **3.4.1. Step 1: Initialize**

During initialization, the CTP Solver attempts to read the database or XML file chosen by the user. If an XML file is chosen, the application checks to make sure three tables exist in the file: config, node, and arc.

If the expected number of tables are not present in the XML file, the CTP Solver will stop execution and print an error message to the user.

If the expected number of tables are present in the XML file, the CTP Solver will read the file and populate the lists of nodes and arcs as well as overwrite the default configuration variables such as Big M and the number of allowed simplex iterations.

#### **3.4.1.1. Set Root Node**

A root node is simply a starting point for the basis tree (described next). From the root node, the path to all other nodes in the network can be traced.

When the CTP Solver reads the node elements from the XML file or database, it first creates an artificial node as the default root node with an id value set as the node count. This node is then inserted into the IList of nodes as the last element so every real node can be referenced with its

natural id/number (assuming nodes are ordered numerically starting at 1 in the original network).

### 3.4.1.2. Create Basis Tree

A basis tree is essentially a feasible minimal spanning tree, or a structure where every node is connected by the minimum required number of arcs and it must be an acyclic directed graph. An example basis tree is shown in Figure 3.2.

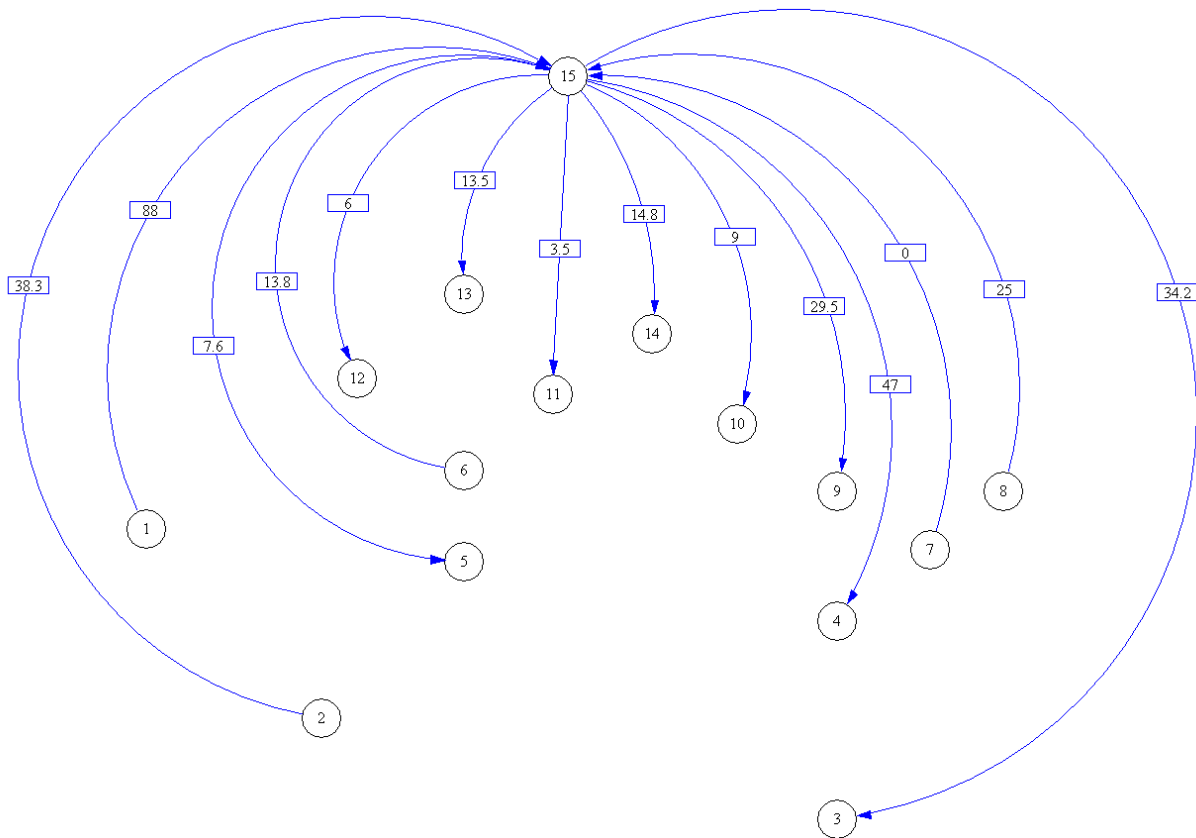


Figure 3.2: Initial Basis Tree of IEEE 14-Bus Test System with Arc Flows.

As a convenience to the user of the application, the CTP Solver automatically creates an initial basis tree to represent a feasible topology of the network, thus removing the need for the user to manually calculate an initial basis and allowing them to focus on and only need knowledge of the specific network values themselves.

Using the generated artificial node as the root, the CTP Solver creates an artificial arc between every real node and the artificial node, forming the initial basis. Each artificial arc in the initial basis is given a cost of the Big M value set by the user in the configuration element, and a flow equal to the absolute value of the node's supply attribute.

Using the Big M method as opposed to the Two Phase method allows the actual arc costs to be used in the first step of the initial basis calculation instead of needing to keep track of the original costs as well as reassign the cost value for every arc in the network.

In addition to initial cost and flow values, each of the artificial arcs is also given an id, starting with the integer data type maximum value and decrementing as needed. This provides a visual differentiation between real and artificial arcs that is easily distinguished at a glance in the results tables.

When determining the head and tail nodes of an artificial arc, the real node's supply value is taken into consideration. If the supply is a positive value, the node is considered a supply node with the real node set as the artificial arc's tail and the artificial root node set as the head. If the supply is a negative value, the node is considered a demand node with the real node set as the artificial arc's head and the artificial root node set as the tail. Transshipment nodes are treated in the same manner as supply nodes.

By directing transshipment nodes toward the root node, the initial basis tree is considered strongly feasible. As such, degenerate arcs, or basic arcs with zero unit flow, can be handled without creating an infinite loop caused by repeatedly iterating through a sequence of degenerate basic feasible solutions corresponding to the same simplex extreme point. [6, pp. 341-343]

#### 3.4.1.3. Determine Node Potentials

In linear programming terms, the node potentials are equivalent to the dual variables. In algorithmic terms, the node potentials are the summed arc costs along the path of any node back to the root node in the basis tree.

Since the initial basis tree essentially consists of a single arc between every real node and the artificial root node, the potential of every node can simply be set using the Big M value specified by the user in the network configuration settings.

The Big M value defined in terms of the CTP Solver is just a number large enough to be considered significantly higher than any existing network values for cost or capacity. While it must be a large value, it cannot be too large as to conflict with the limitations of the data types used (i.e. setting it at the data type's maximum value).

Since the CTP Solver uses the Big M value for artificial arc costs, it could potentially be multiplied by itself as many times as there are number of arcs in the network, however unlikely that may be. This means there must be enough difference between the Big M value and the maximum data type value allowed to ensure a very large node potential can be accurately represented and used in the CTP Solver's calculations.

#### **3.4.2. Step 2: Calculate Reduced Costs**

The reduced cost is the amount the overall total cost of the network could potentially be changed if a given non-basic arc were inserted into the basis. Since the CTP Solver is set up with minimization in mind, the best reduced cost belongs to the arc that will potentially lower the total network cost by the greatest amount.

It should be noted that the CTP Solver could be used for maximization problems by simply using negative cost values. The algorithm will still be

minimizing the optimal solution, but the results can simply be changed from negative to positive values.

#### 3.4.2.1. Non-basic Arcs

By definition, all reduced arc calculations are carried out on non-basic arcs. These arcs are easily represented in an IList object, allowing fast traversal of just those arcs instead of the entire network.

The reduced cost of an arc is calculated using Equation 3.1.

$$R_{ij} = \pi_i - \pi_j - C_{ij}$$

Where:

$R_{ij}$  = Reduced Cost of Arc<sub>ij</sub>  
 $\pi_i$  = Tail Node Potential  
 $\pi_j$  = Head Node Potential  
 $C_{ij}$  = Arc Cost

Equation 3.1: Arc Reduced Cost Calculation.

#### 3.4.2.2. Bidirectional Arcs

The CTP Solver handles bidirectional arcs by simply flipping an arc's head and tail nodes for the reduced cost calculation, shown in Equation 3.2.

$$R'_{ij} = \pi_j - \pi_i - C_{ij}$$

Where:

$R'_{ij}$  = Bidirectional Reduced Cost of Arc<sub>ij</sub>  
 $\pi_i$  = Tail Node Potential  
 $\pi_j$  = Head Node Potential  
 $C_{ij}$  = Arc Cost

Equation 3.2: Bidirectional Arc Reduced Cost Calculation.

If an arc is at its lower bound with no flow, this calculation is done immediately after the normal reduced cost calculation and the two values are then compared. If the bidirectional reduced cost is better than the original reduced cost, the head and tail nodes of the arc are swapped.



It is important to reiterate that non-basic upper bounded arcs and lower bounded arcs with flow cannot be considered bidirectional due to the fact they are already part of the current solution. Allowing these arcs to be treated as bidirectional will often cause net flow violations, rendering the solution infeasible.

By handling bidirectional arcs in this way, the user does not need to duplicate every instance of an arc when the only difference between them is the direction of flow. This simple implementation is actually a very important innovation in directed flow calculations since bidirectional arcs are usually treated as two separate directed arcs. [13, p. 121] The CTP Solver is able to consider the two directions differently even though they are defined only once.

In networks consisting of all bidirectional arcs, the CTP Solver effectively halves the size of the required data file, saving both hard drive space and system memory.

#### 3.4.2.3. Choose Entering Arc

The preferred reduced cost value of a non-basic arc could be positive or negative depending on its bounded flow. If the arc has flow equal to its lower bound, a positive reduced cost is desired. If the arc has flow equal to its capacity, a negative reduced cost is desired.

The reduced cost of every non-basic arc is compared to the best available reduced cost value. When a reduced cost is found to be more attractive, the best available reduced cost arc is replaced by the current arc. This process continues until all non-basic arc reduced costs have been determined and the best available reduced cost arc is chosen.

#### 3.4.2.4. Enforce Lower Bounds

In order to accommodate an attempted enforcement of lower bound flow requirements, any arc with a lower bound value greater than zero with flow less than the lower bound is given priority. The algorithm will force these arcs into the basis and attempt to push as much feasible flow onto them as possible in order to fulfill their lower bound requirements.

With realistic values, this method appears to be sufficient for meeting lower bound flow requirements. However, if the CTP Solver finishes its simplex iterations and determines an optimal solution without meeting all lower bound flow requirements, the application will display a warning message to the user that a lower bound flow violation occurred.

It should be noted that using the lower bound as an initial flow value was implemented as a possible solution to lower bound flow enforcement. Unfortunately, determining an elegant process for guaranteed feasibility was not achieved since it is not always clear which artificial arcs could be updated in conjunction with the lower bounded arc in order to maintain flow balance.

#### 3.4.2.5. Optimality

The CTP Solver assumes optimality until it encounters an attractive entering arc. If no arcs will lower the total network cost when added to the basis tree, the solution is optimal.

#### **3.4.3. Step 3: Create Cycle**

By definition, the basis tree is a connected graph with no cycles. This means there is a path between any two nodes, but not a path from any node to itself. [14, p. 363] When a non-basic arc is added to the basis

tree, a cycle is created and an arc must then be removed to preserve the basis tree's acyclic property.

The process of creating the cycle is the most complex step of the CTP Solver's algorithm. If it were a simple shortest path problem using the arc cost as the arc weight, an algorithm such as Dijkstra's [15] could be used to find an optimal solution. However, the capacitated transshipment problem includes both bounded arcs and directed flow with supply and demand, making it a much more complicated problem.

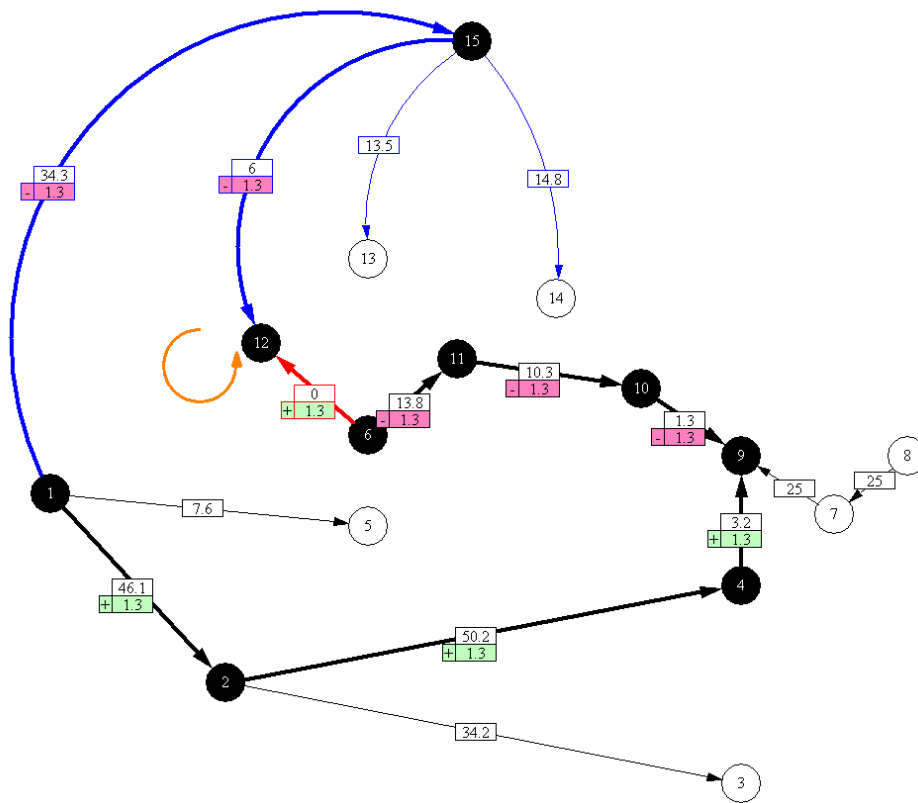


Figure 3.3: IEEE 14-Bus Test System Cycle Iteration 13.

### 3.4.3.1. Add Arcs to Cycle

Determining the entering arc, or the arc added to the basis tree to form a cycle, is a relatively simple process. Traversing that cycle is a bit more complicated.

The algorithm used to create the list of only cycle arcs implements a node depth concept from [10], following the back path from each entering arc's node to the root node of the basis tree.

Using the node depth allows the two back paths to be traversed in pairs during the same iteration, starting at the deepest node in the cycle (highest depth value) and working back up the basis tree until the two back paths meet at the same parent node, or the root node is reached; either of which complete the cycle.

The trick to the CTP Solver's algorithm comes from the need for the head and tail nodes of the entering arc to be handled separately to account for the correct cycle direction modifier: positive one for an arc with flow in the same direction as the entering arc, negative one for an arc with flow in the opposite direction of the entering arc.

By creating a parameterized method for traversing the back path, the same code can be reused with only a few conditional checks for determining the arc's cycle direction.

As the node back paths are followed in this manner, the arc connecting each node and its parent is added to the list of cycle arcs if has not already been added, thus creating the complete cycle.

#### 3.4.3.2. Calculate Maximum Feasible Flow Change

As each arc is added to the cycle, its maximum feasible flow change is calculated based on the arc's direction in relation to the cycle created by the entering arc. This value, represented by the Greek letter theta, is the largest amount of flow units that could be added or subtracted from a same- or opposite-cycle direction arc, respectively, without violating the arc's flow capacity or lower bound requirement.

Using the theta value, the CTP Solver's algorithm ensures that each simplex iteration moves the current basis tree as close to the optimal solution as is feasibly possible.

For same-cycle direction arcs, the value of theta is simply the difference between the arc's capacity and its current unit flow. For opposite-cycle direction arcs, the value of theta is calculated as the difference between the arc's current unit flow and its lower bound.

#### 3.4.3.3. Choose Leaving Arc

Once an arc's theta value has been determined, it is compared against the current minimum theta value for the cycle. To maintain feasibility when an arc is removed from the cycle, the smallest theta value from all the cycle arcs must be used to ensure no capacity or lower bound violations occur.

The cycle's minimum theta value can only be changed if the current arc's theta value is strictly less than the cycle's overall minimum, or if the arc is artificial. These two possible theta updating conditions prevent infinite cycles due to degeneracy and force artificial arcs out of the basis, respectively.

#### 3.4.3.4. Update Cycle Flows

Once the leaving arc has been determined, the cycle is iterated a final time in order to add or remove theta units of flow to its arcs. Using an arc's direction property, flow is added to same direction cycle arcs and subtracted from opposite direction cycle arcs.

By following the cycle direction, the solution's feasibility is ensured since arc limits are not capable of being violated by adding or subtracting too many flow units.

#### 3.4.3.5. Degeneracy

Recall from the initial basis creation that a degenerate arc is one with a unit flow of zero. Degenerate arcs can cause infinite loops and must be handled properly to avoid such problematic outcomes. The initial basis is created to be strongly feasible and the CTP Solver needs to maintain that status.

Using the node depth method described in the cycle creation, the lowest, or deepest degenerate arc can be chosen to leave the basis, preserving a strongly feasible basis. [10]

This is accomplished by the algorithm's tie-breaking conditional check that occurs when determining the minimum cycle theta value. Since the algorithm starts at the deepest cycle arc, a simple comparison can be made between the current theta value and the cycle's minimum value and only change the value of theta if the former is less than the latter, thus always choosing the deepest cycle arc.

#### **3.4.4. Step 4: Update Basis**

Updating the basis involves a recursive method, or a method that calls itself, starting from the root node as the top of the basis tree and working down one node level at a time until all nodes and arcs of the basis have been updated.

Possible errors could result in the allocation of the system's memory [16] during the recursive process, however the node and arc structures used in the method are already stored completely in memory using the IList structures. So if memory allocation is an issue, it would likely occur before the recursive process even begins.

A possible optimization, discussed later, would be to update only cycle nodes and arcs instead of the entire basis tree. But the use of recursion

through the full basis tree was chosen here due to its simplification of the algorithmic process, essentially implementing an easily comprehended depth-first search [17, p. 85]. The search space for each iteration is the size of a spanning tree, or one less than the number of nodes in the network [13, p. 236].

During each recursive iteration of the basis update method, the CTP Solver uses a LINQ query to find all child nodes of the current node. Then for each child node, the method calls itself to find that node's child nodes. This process repeats until the entire basis tree has been traversed and the node potential and depth values have all been updated.

Once the basis tree has been updated, it is ready to be used for the next simplex iteration unless it is already optimal.

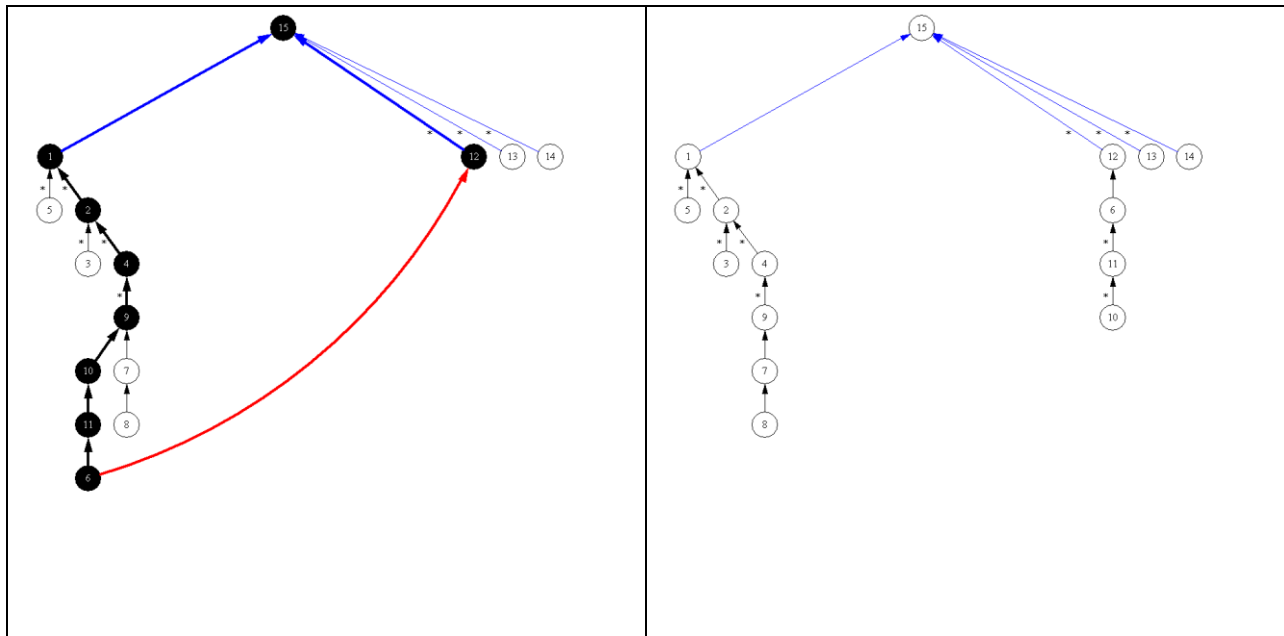


Figure 3.4: IEEE 14-Bus Test System Basis Update 13.

### **3.4.5. Step 5: Repeat Steps 2-4 Until Optimal**

The CTP Solver assumes optimality until the non-basic arc reduced cost values have been calculated in Step 2. If adding a non-basic arc to the basis will lower the overall total cost of the network solution, the

optimality flag is set to false and the CTP Solver executes another simplex iteration, continuing through Step 3 and Step 4.

Once the reduced cost calculation step determines there are no non-basic arcs that should become entering arcs, the process is complete and the solution is optimal.

### **3.5. Output**

The CTP Solver provides the user with all available information about the resulting network in addition to the step-by-step simplex iterations and cycle debugging log. The user is also shown a link to the original network file and optimal solution available for download in either XML or .CSV format.

#### **3.5.1. Miscellaneous Information**

Various information about the results is displayed to the user before any other data. First are links to the network files, including the original network (if the source was an XML file) as well as downloadable .CSV and XML files of the optimal solution.

After the network file links, the optimal network cost, number of simplex iterations, node count, basic arc count, non-basic arcs with flow count, and execution time are all displayed.

Used in conjunction with the optimal network table, the user is able to quickly understand the results of the CTP Solver and download the information for analysis or importing into other systems.

A screenshot of the miscellaneous information for the IEEE 14 bus test system using distance as a cost measure is shown in Figure 3.5.



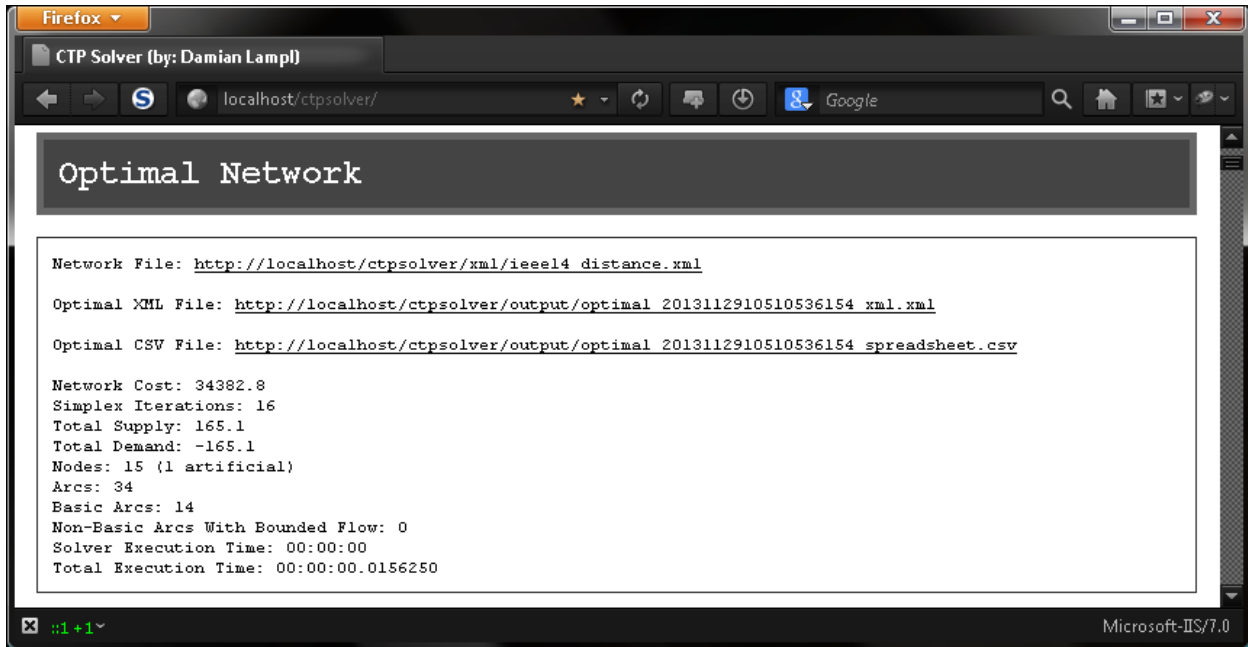


Figure 3.5: CTP Solver Optimal Network Miscellaneous Information (Screenshot, Light CSS).

### 3.5.2. Optimal Solution

After the miscellaneous results information, the optimal network table is displayed on the results page, showing the optimal network topology via the list of arcs. Since the node information is not required for reconstructing the optimal network, the list of nodes is omitted.

The table showing the optimal solution includes each arc's id, tail node, head node, cost, capacity, lower bound, flow, capacity minus flow, reduced cost, basis order, and whether or not it is a basic arc or an arc with bounded flow (denoted as "non-basic"). All non-basic arcs without bounded flow are simply displayed with a hyphen for the basis value.

Artificial arcs are included in the final optimal display table; however, they are visually separated from the real network arcs and are not included in the XML or .CSV exports.

A screenshot of the optimal solution for the IEEE 14 bus test system using distance as a cost measure is shown in Figure 3.6.

OPTIMAL NETWORK	Tail	Head	Cost	Capacity	Lower Bound	Flow	Capacity - Flow	Reduced Cost	Order	Basis
2147483647	1	15	1000	M	0	0.0	M	0	0	-
2147483646	2	15	1000	M	0	0.0	M	0	0	-
2147483645	15	3	1000	M	0	0.0	M	0	0	-
2147483644	15	4	1000	M	0	0.0	M	0	0	-
2147483643	15	5	1000	M	0	0.0	M	0	0	-
2147483642	6	15	1000	M	0	0.0	M	0	0	-
2147483641	7	15	1000	M	0	0	M	0	0	-
2147483640	8	15	1000	M	0	0	M	0	0	-
2147483639	15	9	1000	M	0	0.0	M	0	0	-
2147483638	15	10	1000	M	0	0.0	M	0	0	-
2147483637	15	11	1000	M	0	0.0	M	0	0	-
2147483636	15	12	1000	M	0	0.0	M	0	0	-
2147483635	15	13	1000	M	0	0.0	M	0	1	basic
2147483634	15	14	1000	M	0	0.0	M	0	0	-
OPTIMAL NETWORK	Tail	Head	Cost	Capacity	Lower Bound	Flow	Capacity - Flow	Reduced Cost	Order	Basis
1	1	2	112	M	0	62.2	937.8	0	5	basic
3	2	3	124	M	0	34.2	965.8	0	6	basic
4	2	4	167	M	0	66.3	933.7	0	7	basic
6	3	4	172	M	0	0	M	-129	0	-
2	1	5	144	M	0	25.8	974.2	0	4	basic
5	2	5	128	M	0	0.0	M	-96	0	-
7	5	4	148	M	0	0	M	-13	0	-
10	5	6	50	M	0	18.2	981.8	0	3	basic
8	7	4	50	M	0	0	M	-30	0	-
14	8	7	114	M	0	25	975	0	10	basic
9	4	9	0	M	0	19.3	980.7	0	8	basic
15	7	9	20	M	0	25	975	0	9	basic
16	10	9	34	M	0	0.0	M	-7	0	-
11	6	11	50	M	0	12.5	987.5	0	12	basic
18	11	10	8	M	0	9.0	991.0	0	13	basic
12	6	12	94	M	0	6.0	994.0	0	14	basic
13	6	13	151	M	0	13.5	986.5	0	2	basic
19	12	13	86	M	0	0	M	-29	0	-
17	9	14	56	M	0	14.8	985.2	0	11	basic
20	14	13	56	M	0	0	M	-46	0	-

Figure 3.6: CTP Solver Optimal Network Arc Information (Screenshot, Light CSS).

### **3.5.3. Simplex Iterations**

The information for each individual simplex iteration can be shown if the user so chooses in the configuration options using the "showSimplexIterations" attribute.

When shown, every iteration is given a separate expandable block of information detailing a snapshot of the network. In addition to the same information displayed for the optimal network list of arcs, each cycle, entering and leaving arc, and an individual table for the basis tree are shown.

Showing the data for each simplex iteration can be very useful in debugging as a way to step through the algorithm's process to follow every decision made for verification purposes. The first iteration is the network as it is provided to the CTP Solver, with successive iterations showing the evolving network as the algorithm progresses.

As an example of the information provided by the CTP Solver, screenshots of the thirteenth simplex iteration for the IEEE 14 bus test system using distance as a cost measure are shown in Figures 3.7, 3.8, and 3.9 below.

Firefox

CTP Solver (by: Damian Lamp)

localhost/ctpsolver/

Simplex Iteration: 13

Node	i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	-
<b>Head</b>	<b>H</b>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	-
<b>Parent</b>	<b>P</b>	15	-1	-2	-2	-1	12	9	7	-4	-11	-6	-15	-15	-15	0	-
<b>Preorder Thread</b>	<b>IT</b>	2	3	4	9	0	11	8	5	7	1	10	6	12	13	14	15
<b>Supply</b>	<b>b</b>	88	38.3	-34.2	-47	-7.6	13.8	0	25	-29.5	-9	-3.5	-6	-13.5	-14.8	0	-
<b>Depth</b>	<b>D</b>	1	2	3	3	2	2	5	6	4	4	3	1	1	1	0	-
<b>Potential</b>	<b><math>\pi</math></b>	1000	888	764	721	856	-906	741	855	721	-964	-956	-1000	-1000	-1000	0	-

Microsoft-UIS/7.0

Figure 3.7: CTP Solver IEEE 14-Bus System Simplex Iteration 13 - Node Details (Screenshot, Light CSS).

ALL NETWORK ARCS	Tail	Head	Cost	Capacity	Lower Bound	Flow	Capacity - Flow	Reduced Cost	Order	Basis
2147483647	1	15	1000	M	0	33.0	967.0	0	7	basic
2147483646	2	15	1000	M	0	0.0	M	0	0	-
2147483645	15	3	1000	M	0	0.0	M	0	0	-
2147483644	15	4	1000	M	0	0.0	M	0	0	-
2147483643	15	5	1000	M	0	0.0	M	0	0	-
2147483642	6	15	1000	M	0	0.0	M	0	0	-
2147483641	7	15	1000	M	0	0	M	0	0	-
2147483640	8	15	1000	M	0	0	M	0	0	-
2147483639	15	9	1000	M	0	0.0	M	0	0	-
2147483638	15	10	1000	M	0	0.0	M	0	0	-
2147483637	15	11	1000	M	0	0.0	M	0	0	-
2147483636	15	12	1000	M	0	4.7	995.3	0	3	basic
2147483635	15	13	1000	M	0	13.5	0	0	2	basic
2147483634	15	14	1000	M	0	14.8	0	0	1	basic
1	1	2	112	M	0	47.4	952.6	0	8	basic
3	2	3	124	M	0	34.2	965.8	0	9	basic
4	2	4	167	M	0	51.5	948.5	0	10	basic
6	3	4	172	M	0	0	M	-129	0	-
2	1	5	144	M	0	7.6	992.4	0	14	basic
5	2	5	128	M	0	0.0	M	-96	0	-
7	5	4	148	M	0	0	M	-13	0	-
10	5	6	50	M	0	0.0	M	-7	0	-
8	7	4	50	M	0	0	M	-30	0	-
14	8	7	114	M	0	25	975	0	13	basic
9	4	9	0	M	0	4.5	995.5	0	11	basic
15	7	9	20	M	0	25	975	0	12	basic
16	10	9	34	M	0	0.0	M	0	0	-
11	6	11	50	M	0	12.5	987.5	0	5	basic
18	11	10	8	M	0	9.0	991.0	0	6	basic
12	6	12	94	M	0	1.3	998.7	1719	4	basic
13	6	13	151	M	0	0	M	1662	0	-
19	12	13	86	M	0	0	M	-86	0	-
17	9	14	56	M	0	0	M	1665	0	-
20	13	14	56	M	0	0	M	-56	0	-

Figure 3.8: CTP Solver IEEE 14-Bus System Simplex Iteration 13 - Full Network Results (Screenshot, Light CSS).

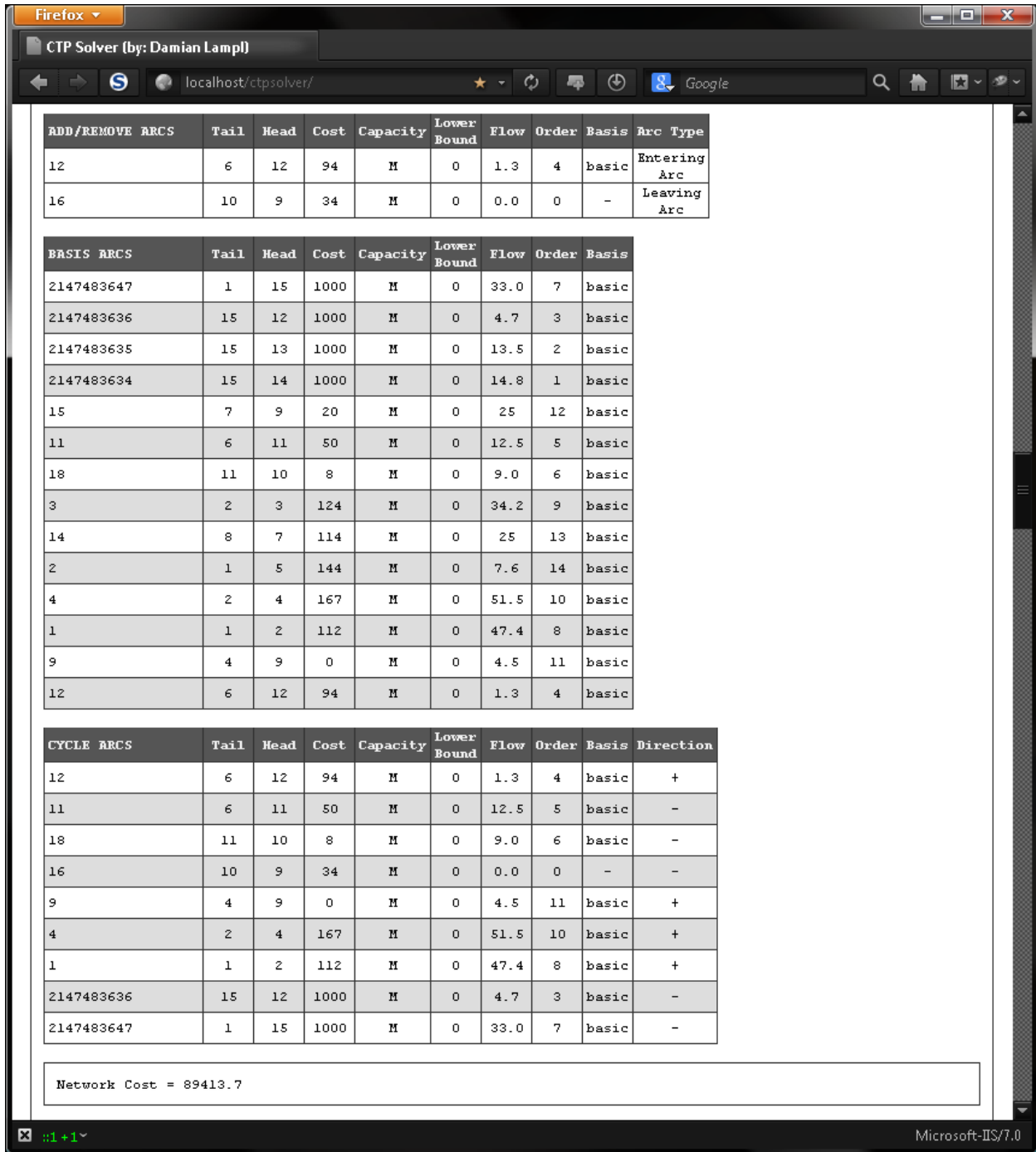


Figure 3.9: CTP Solver IEEE 14-Bus System Simplex Iteration 13 - Entering/Leaving Arcs, Basis, Cycle, and Current Network Cost (Screenshot, Light CSS).

### 3.5.4. Debugging Log

The debugging log includes the cycle created by each simplex iteration. As with the simplex iterations, the cycle debugging log can be toggled by the

user in the network file's configuration element using the "showDebuggingLog" attribute.

Each cycle begins with its entering arc and shows each cycle arc's direction in relation to that entering arc, along with its maximum feasible flow change.

Just like the simplex iterations, the cycle debugging log can be valuable information for tracing through the CTP Solver's algorithmic process for verification.

### **3.6. Limitations and Modifications**

Not all networks are guaranteed to have a feasible solution. The CTP Solver handles these networks by providing warning messages to the user prior to the display of the best possible solution the application was able to obtain.

These warning messages alert the user to infeasibilities such as lower bound flow violations, artificial arcs unable to be removed from the basis, and net flow violations on any nodes.

#### **3.6.1. Performance Gains**

The CTP Solver displays a lot of information to the user. However, some users may not be interested in the output generated for every simplex iteration or the debugging log showing each cycle. With these users in mind, the debugging and individual simplex iterations can simply be turned off by setting the respective variables in the configuration element of the network file.

Since each simplex iteration and cycle traversal generates data proportional to the network size that must be displayed during every new iteration, significant gains in execution speed can be achieved by choosing to not show this information for larger networks.

For example, a performance increase greater than an order of magnitude was observed on a test network with just 30 nodes and 55 arcs by hiding only the individual simplex iterations. That performance was doubled when the debugging log was also hidden. Full performance details are available in the results.

### **3.7. Network Generator**

In order to test multiple networks of varying sizes and values, a network generator was developed to accelerate the process of creating XML files for use in the CTP Solver.

The network generator is capable of reading the existing IEEE test system text files and generating some values such as arc costs, as well as creating completely random networks with user-defined topology values and limits.

In addition to exporting XML files for the CTP Solver, the network generator also exports data files for use in AMPL and SAS. Automating the creation of these additional files made comparisons between the CTP Solver, AMPL, and SAS much easier to conduct while also removing any user error caused by manual editing.

#### **3.7.1. IEEE Test Files**

The various IEEE test system files included basic network topology that was easily read and exported to XML for use in the CTP Solver.

However, some values either required specific calculations based on the entire system or, in the case of values such as arc costs, information was not directly included in the standard files.

In some cases, the calculations for the true or exact values was deemed outside the scope of the CTP Solver's purpose, leading to the generation of



estimated or sometimes even completely random values within a user-defined minimum and maximum range.

In each test system, the process for determining the network values is clearly described in cases where they were generated or estimated. For all test systems, the entire set of network values used for testing is provided in the XML files on the CTP Solver website, allowing easy comparisons against different methods.

#### 3.7.1.1. Formatting Discrepancies

Despite a recommended standard format, not all of the IEEE test system files were able to be read by the network generator in the exact same way. To accommodate the discrepancies and allow easier imports of other currently unused as well as possible future test networks, a generic network file information class was created to allow different formatting practices to be handled in an efficient manner.

Some example settings for individual network files include the start and stop line index of node IDs, names, and supply values.

##### 3.7.1.1.1. Duplicate Arcs

The CTP Solver requires a single arc connecting any pair of nodes. The 57-, 118-, and 300-bus test system network files all had duplicate arc listings, causing errors when used by the CTP Solver.

In each instance, an arc was disregarded if it had the same head and tail nodes (in any combination) as an existing network arc. This means only the first instance of a given arc was added to the final network topology.

#### 3.7.1.1.2. Sequential Node ID Requirement

In order to capitalize on computational advantages provided by IList objects, the lists of nodes and arcs both require sequential IDs. In the 300-bus test system, the node IDs are not sequential.

To make the data useful in the CTP Solver, the network generator creates a node ID mapping, allowing the correct nodes to be referenced in the arc definitions while reordering the node IDs into sequential values.

#### 3.7.1.2. Supply and Demand

The supply values are taken directly from the IEEE test system files using the Base KV (F) column for the 14-, 30-, and 57-bus systems, and the Generation MVAR (F) column for the 118- and 300-bus systems. If the column was a non-zero value, the absolute value of that number was used as the node's supply. If a node has zero maximum flow in due to no incoming arcs, it is provided a random supply value based on the user settings.

The total supply for the network is summed and used as available demand since the supply must equal the demand for the CTP Solver to calculate an optimal solution.

After the transshipment nodes are removed from the list of demand nodes, a random value is generated between a user-defined minimum and the average of the available demand, defined as the available demand divided by the number of remaining demand nodes. The negative of this value is used as the node's supply (recall that the CTP Solver represents node demand with a negative supply value).

The last demand node is given the remaining available demand, ensuring the total supply equals the total demand for the entire network. If the value of the final demand node exceeds its maximum total flow in, the network will be infeasible. However, the capacities of its incoming arcs can simply

be adjusted manually, or the entire network can be quickly re-generated with new random values.

### **3.7.2. Generic Networks**

In addition to reading the standard IEEE test system files, the network generator was built with pseudorandom generic network generation in mind.

When creating a generic network, the user is able to set a range of minimum and maximum values for:

1. Node Supply
2. Arc Capacity
3. Arc Lower Bound
4. Arc Cost

In addition, the user can also set values for the total network supply and a lower bound frequency threshold, defined as an integer value from 1-100 essentially acting as a percentage for approximately how often the user would like a lower bound value to occur for network arcs.

#### **3.7.2.1. Realistic Networks**

According to Wang, et al. [18], realistic smart grid network topologies share some characteristics with small-world network models; primarily a sparse connectivity with low average nodal degree that does not scale with the network size.

With that in mind, the network generator was set up to generate topologies with no nodal degree greater than seven (although this setting is customizable by the user). Arcs are created by looping through the nodes, checking the degree, and randomly connecting up to seven nodes on either side of the current node. This ensures the neighborhood connectivity characteristics of the small-world network model.

During arc generation, the network generator modifies the algorithm proposed in [18] by introducing what amounts to a genetic algorithm mutation [17, p. 128], giving each node a one-percent chance to connect to a node outside its immediate neighborhood.

Limiting the node degree as well as allowing a small chance for connections outside of a node's neighborhood in this manner allows the generated network topologies to be sufficiently realistic for testing purposes.

## 4. RESULTS

### 4.1. Software Comparisons

The CTP Solver uses a customized simplex algorithm, implemented in ASP.NET C#. The results of multiple test networks were compared to two separate optimization software programs: AMPL and SAS.

#### 4.1.1. AMPL

AMPL is a comprehensive and powerful algebraic modeling language for linear and nonlinear optimization problems, developed at Bell Laboratories. [19]

One of the primary benefits of AMPL is its separation between the model and data files, allowing the user to use the same model on multiple datasets. The user is expected to learn AMPL's syntax to create their own models for specific applications.

AMPL also allows the user to choose from many different custom solution solvers. The solver chosen for the test networks was LPSOLVE, an open source simplex solver.

The version of AMPL used on the test networks was AMPL Student Version 20100715 (MS VC++ 6.0). The LPSOLVE solver version 4.0.1.0 was used since it allowed the highest number of variables and constraints with the student version of AMPL.

Due to these software limitations, however, the IEEE 300-Bus Test System could not be solved using the student version of AMPL since it had too many variables and constraints.

#### **4.1.2. SAS**

SAS is a collection of software solutions used for solving complex business problems based on three key capabilities: information management, analytics, and business intelligence. [20]

As with AMPL, SAS is a powerful software tool with the ability to separate a problem model from its data. SAS also requires its users to learn its programming language syntax in order to create their own models.

The built-in SAS method used for the test networks was the NETFLOW procedure. Unlike AMPL's LPSOLVE solver and the CTP Solver, the NETFLOW procedure uses the interior point algorithm [21] instead of the simplex algorithm. It also uses the "good path" method described in "Algorithms for Networking Programming" by J. Kennington and R. V. Helgason. [22]

Version 9.3 of the X64\_VSPRO platform version of the SAS software was used for the test networks.

#### **4.1.3. CTP Solver**

Due to the design goals of the CTP Solver's implementation, it is able to offer some key benefits to its users not provided by AMPL or SAS. Most of these benefits are directly related to the ease of use design goal of the CTP Solver, focusing on simplicity for its users.

##### **4.1.3.1. Modeling**

While incredibly robust and capable solutions, both AMPL and SAS require their users to understand how to model their problems in order to understand and utilize the solutions. This allows many more different kinds of problems to be solved, but the learning curve may be too steep for most users due to each software application having its own syntax.

Since the CTP Solver abstracts the user from the modeling process, it can simply be used with properly formatted XML files or a database.

#### 4.1.3.2. Software Installation

The installation process for SAS in particular can be an overwhelming experience for typical users, requiring both the SAS software as well as Java runtime dependencies. It includes many different business analytics, intelligence, and information management solutions, making it an extremely complex process before even using the software.

AMPL does not require installation, but it does require downloading and extraction, as well as user knowledge of the program's file structure in order to find the data and model files.

Since the CTP Solver is a web application, a user simply needs a browser in order to access and utilize it, making it more accessible than either AMPL or SAS.

#### 4.1.3.3. Output

Both AMPL and SAS produce simplified results by default, with SAS more closely resembling the CTP Solver's default table output.

However, the CTP Solver also exports its results to XML and .CSV data files by default, or directly to the database if the input network is from a database source. Customized output requires more user effort in both AMPL and SAS than is required by the CTP Solver.

## **4.2. Reading the CTP Solver Results**

The CTP Solver displays a lot of information to the user, but does so in an organized manner.

#### 4.2.1. Optimal Network

The first results shown are miscellaneous details about the solution, including the optimal output files in .XML and .CSV format, the minimized cost of the network, the number of simplex iterations performed, total network supply and demand, the node count, the arc count, the number of basic arcs in the solution, the number of non-basic arcs with bounded flow in the solution, and the execution time.

Displaying this information first allows the user to quickly view the important details of the optimal solution, including any errors that might have occurred during the CTP Solver's progress.

After the optimal results summary, the entire solution network is displayed, including non-basic arcs. Normally the non-basic arc information could be discarded, but in the event of any errors, having them displayed could give the user some insights into reasons why the network might be infeasible and where possible changes could be made in an effort to modify the network into a feasible topology.

When the network file is read into memory, the arcs are sorted in increasing head node format. However, if an arc is bidirectional and its head and tail nodes have been reversed, the result will display out of sequence.

To make the real arcs of the optimal solution easier to read, the table header is repeated every twenty rows. The header is not repeated for the artificial arcs since only one of them is of any importance if the network is feasible. This also serves the purpose of making it easier to see where the real network arcs are separated from the artificial arcs.

Most of the information in the results table is self-explanatory, including the tail node ID (Tail), head node ID (Head), cost, capacity, lower bound, flow, capacity minus flow, and reduced cost of every arc. The first



column under the "Optimal Network" heading is simply the arc ID. The "Order" of the arc is its order in the basis tree. This column is mostly just useful for debugging purposes. The "Basis" column denotes basic arcs (basic) and non-basic arcs with bounded flow (non-basic) to help the user identify all arcs with flow. Arcs with a value of "-" in the Basis column have a flow of zero and are not included in the optimal solution network.

#### **4.2.2. Simplex Iterations**

Detailed information can be displayed for every simplex iteration the CTP Solver calculates. This can be helpful in determining the exact process followed by the algorithm in order to ensure the solver's accuracy and help debug any errors encountered.

Each iteration is numbered, with the first iteration, zero, representing the network as it was read from the data file or database. Individual iterations can be toggled to show/hide the information presented, allowing the user to quickly access a specific iteration and view its results.

The information for each iteration is broken down into six tables: nodes, current network arc values, entering and leaving arcs, basis arcs, cycle arcs, and network cost.

Because the CTP Solver is displaying so much information, showing each simplex iteration is not recommended for large networks due to the significantly larger amount of time needed to write the information to the page.

##### **4.2.2.1. Nodes**

The information for each node at the current iteration is shown, including values for a node's parent, preorder thread, supply, depth, and potential.

If a node's parent is negative, that means the arc is reflected in the basis tree. The second-to-last node is the artificial root node, automatically generated for constructing an initial basis.

The preorder thread can be followed starting with the numeric value in the final column. That value is the root node of the basis tree. From the root node, each successive node can be followed using the preorder thread value until a value of zero is reached, signifying the last node in the basis tree.

The supply of each node is shown with a negative value denoting demand. The depth is the node's level in the basis tree, representing the number of parents in the node's back path to the root node. The potential of a node is its summed cost along its back path to the root node, with reflected arcs subtracted from its sum total.

#### 4.2.2.2. Current Network Values

After the information describing the nodes, all network arcs are displayed, including artificial arcs. This information is the same as shown in the optimal network table with the best reduced cost, representing the entering arc, highlighted. If multiple arcs have the same best reduced cost, the first arc encountered is chosen as the entering arc.

#### 4.2.2.3. Entering and Leaving Arcs

The entering and leaving arc are displayed in a separate table to make them easier to specifically distinguish from other arcs.

#### 4.2.2.4. Basis

Each arc of the basis is also displayed in a separate table from the full network, again for the sole purpose of making it easier to follow without piecing everything together using the full network.

#### 4.2.2.5. Cycle Arcs

As with the entering and leaving arcs as well as the basis arcs, the individual cycle arcs along with their directions are displayed in a separate table in order to more easily follow.

If an arc's direction is denoted with a plus sign (+), it means the arc follows the same direction as the entering arc. If its direction is denoted with a minus sign (-), it means the arc follows the opposite direction as the entering arc.

#### 4.2.2.6. Network Cost

The total network cost of all basic and non-basic arcs with flow is shown, calculated as the sum total of each arc's flow units multiplied by its cost for all arcs with flow.

#### **4.2.3. Cycle Debugging Log**

The cycle debugging log shows specific cycle details not displayed in the cycle arc tables of each simplex iteration, including the updated theta value (maximum feasible flow change), maximum feasible flow change for each cycle arc, and the positive or negative theta amount updated for each cycle arc's flow.

Displaying all of this additional information as well as just the cycle arcs in the simplex iterations allows the user more information for debugging and following each step of the algorithm's progress.

#### **4.2.4. Displayed Results Comparison**

All three applications used in testing networks display information to the user in their own way. The primary two aspects of particular interest are the display of the optimal solution value along with key diagnostics

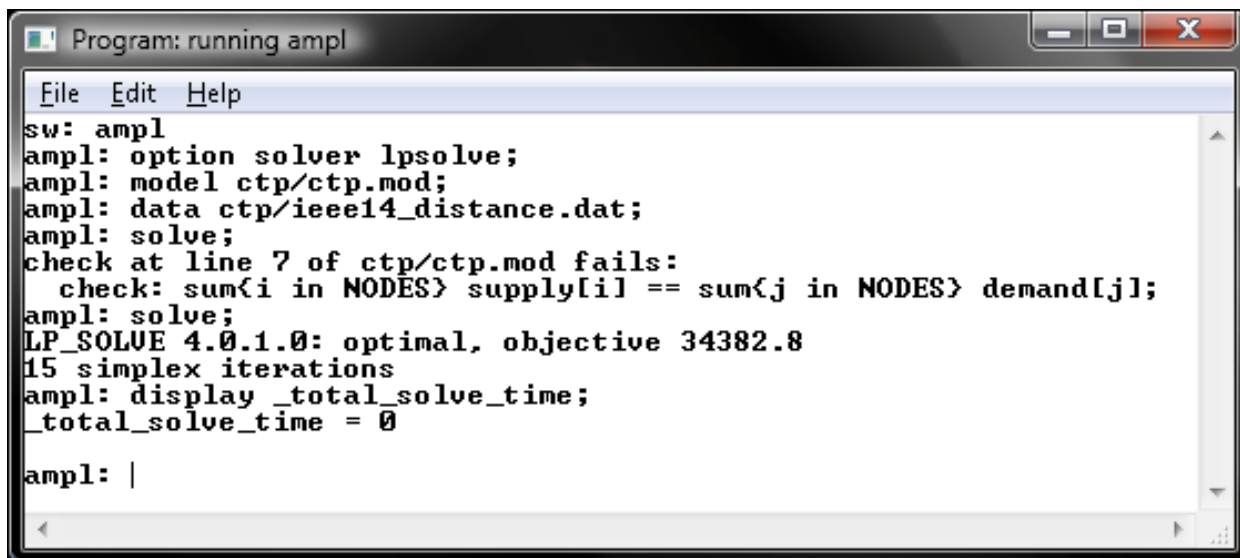
information regarding the solver's performance, and the topology of the resulting optimal network including flows.

#### 4.2.4.1. Optimal Display and Diagnostics Comparison

Both SAS and the CTP Solver do a relatively good job of providing the user with important detailed information in an easy manner, while AMPL requires a bit more effort from the user to find some of the relevant data.

The CTP Solver displays this important information immediately before the optimal network topology, allowing the user to quickly determine key aspects of the results. SAS also displays its information in a way that is easy for the user to access, using its log output window.

But when using AMPL, the user must specifically print diagnostics they are interested in viewing. This requirement subjectively makes the diagnostics display in AMPL a bit more cumbersome than SAS or the CTP Solver since the user must read through the documentation and become familiar with the relevant variables and how they are used in AMPL.



```
Program: running ampl
File Edit Help
sw: ampl
ampl: option solver lpsolve;
ampl: model ctp/ctp.mod;
ampl: data ctp/ieee14_distance.dat;
ampl: solve;
check at line 7 of ctp/ctp.mod fails:
  check: sum<i in NODES> supply[i] == sum<j in NODES> demand[j];
ampl: solve;
LP SOLVE 4.0.1.0: optimal, objective 34382.8
15 simplex iterations
ampl: display _total_solve_time;
_total_solve_time = 0

ampl: |
```

Figure 4.1: AMPL Optimal and Diagnostics Display (Screenshot).

```

Log - (Untitled)
73      run;

NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 9 .
NOTE: Total supply= 165.1 , total demand= 165.1 .
NOTE: Number of arcs= 40 .
NOTE: Number of iterations performed (neglecting any constraints)= 21 .
NOTE: Of these, 4 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= 34382.8 .
NOTE: The data set WORK.ARC_SAV has 40 observations and 13 variables.

NOTE: PROCEDURE NETFLOW used (Total process time):
      real time          0.68 seconds
      cpu time           0.26 seconds

74      proc print label;
75          var tail head cost capacity lowerbound _supply_ _demand_
76              _flow_ _fcost_ _rcost_;
77          sum _fcost_;
78      run;

NOTE: There were 40 observations read from the data set WORK.ARC_SAV.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.10 seconds
      cpu time           0.03 seconds

```

Figure 4.2: SAS Optimal and Diagnostics Display (Screenshot).

```

Firefox
CTP Solver (by: Damian Lamp)
localhost/ctpsolver/
Optimal Network

Network File: http://localhost/ctpsolver/xml/ieeel4\_distance.xml
Optimal XML File: http://localhost/ctpsolver/output/optimal\_2013112910510536154\_xml.xml
Optimal CSV File: http://localhost/ctpsolver/output/optimal\_2013112910510536154\_spreadsheet.csv

Network Cost: 34382.8
Simplex Iterations: 16
Total Supply: 165.1
Total Demand: -165.1
Nodes: 15 (1 artificial)
Arcs: 34
Basic Arcs: 14
Non-Basic Arcs With Bounded Flow: 0
Solver Execution Time: 00:00:00
Total Execution Time: 00:00:00.0156250

```

Figure 4.3: CTP Solver Optimal and Diagnostics Display (Screenshot, Light CSS).

#### 4.2.4.2. Optimal Network Comparison

In addition to important information about the results, all three applications also provide a representation of the optimal network with flows along the arcs.

Here again, SAS and the CTP Solver have a bit of an edge on AMPL in their simplicity and automatic display of the results to an HTML table. In AMPL, the user is required to manually display the results of the optimal network matrix.

However, both AMPL and the CTP Solver display the optimal network in a logical order, whereas SAS prints the network arcs in an apparent random order. The CTP Solver orders arcs in increasing head format; for example, it starts with all arcs directed toward Node 1, then all arcs directed toward Node 2, and so on. Depending on whether or not the arc was flipped due to its bidirectional counterpart being the best for the optimal solution, the head node may be flipped in the display, but they are all shown based on their original orientation.

The CTP Solver also separates itself from AMPL and SAS with a few key features, including exporting the resulting network to both XML and CSV for easy use in other applications and portability among different systems.

One other nice feature of the CTP Solver is the inclusion of repeating table headers every 20 rows after the artificial arcs, allowing the user to easily see which information is in each table cell at a glance as opposed to scrolling all the way back up to the top as required by SAS.

The CTP Solver also differentiates between basic and non-basic arcs with flow, providing the user with more detailed information when bounded arcs are included in the solution.

```

Program: running ampl
File Edit Help
sw: ampl
ampl: option solver lpsolve;
ampl: model ctp/ctp.mod;
ampl: data ctp/ieee14_distance.dat;
ampl: solve;
check at line 7 of ctp/ctp.mod fails:
  check: sum{i in NODES} supply[i] == sum{j in NODES} demand[j];
ampl: solve;
LP SOLVE 4.0.1.0: optimal, objective 34382.8
15 simplex iterations
ampl: display _total_solve_time;
_total_solve_time = 0

ampl: display Flow;
Flow [*,*]
:      1      2      3      4      5      6      7      8      9      10     11     12     13 :=
1      .      62.2      .      .      25.8      .      .      .      .      .      .      .
2      0      .      34.2      66.3      0      .      .      .      .      .      .      .
3      .      0      .      0      .      .      .      .      .      .      .      .
4      .      0      0      .      0      .      .      .      19.3      .      .      .
5      0      0      .      0      .      18.2      .      .      .      .      .      .
6      .      .      .      .      0      .      .      0      .      .      12.5      6      13.5
7      .      .      .      0      .      .      .      .      25      .      .      .
8      .      .      .      .      .      .      25      0      .      .      .      .
9      .      .      .      0      .      .      .      .      .      0      .      .
10     .      .      .      .      .      .      .      .      0      .      0      .
11     .      .      .      .      .      0      .      .      .      9      .      .
12     .      .      .      .      .      0      .      .      .      .      .      0
13     .      .      .      .      .      0      .      .      .      .      .      0
14     .      .      .      .      .      .      .      .      0      .      .      0

:      14      :=
9      14.8
13     0
;

ampl: |

```

Figure 4.4: AMPL Optimal Network (Screenshot).

Results Viewer - SAS Output

**The SAS System**

Obs	tail	head	cost	capacity	lowerbound	Supply of tail node.	Demand of head node.	Arc flow or Nonarc value.	Arc flow*cost, Nonarc value*objfn coef.	Arc or Nonarc reduced cost.
1	2	1	112	1000	0	38.3	.	0.0	0.0	224
2	5	1	144	1000	0	.	.	0.0	0.0	288
3	9	10	34	1000	0	.	9.0	0.0	0.0	61
4	11	10	8	1000	0	.	9.0	9.0	72.0	.
5	6	11	50	1000	0	13.8	3.5	12.5	625.0	.
6	10	11	8	1000	0	.	3.5	0.0	0.0	16
7	6	12	94	1000	0	13.8	6.0	6.0	564.0	.
8	13	12	86	1000	0	.	6.0	0.0	0.0	143
9	6	13	151	1000	0	13.8	13.5	13.5	2038.5	.
10	12	13	86	1000	0	.	13.5	0.0	0.0	29
11	14	13	56	1000	0	.	13.5	0.0	0.0	46
12	9	14	56	1000	0	.	14.8	14.8	828.8	.
13	13	14	56	1000	0	.	14.8	0.0	0.0	66
14	1	2	112	1000	0	88.0	.	62.2	6966.4	.
15	3	2	124	1000	0	.	.	0.0	0.0	248
16	4	2	167	1000	0	.	.	0.0	0.0	334
17	5	2	128	1000	0	.	.	0.0	0.0	160
18	2	3	124	1000	0	38.3	34.2	34.2	4240.8	.
19	4	3	172	1000	0	.	34.2	0.0	0.0	215
20	2	4	167	1000	0	38.3	47.0	66.3	11072.1	.
21	3	4	172	1000	0	.	47.0	0.0	0.0	129
22	5	4	148	1000	0	.	47.0	0.0	0.0	13
23	7	4	50	1000	0	.	47.0	0.0	0.0	30
24	9	4	0	1000	0	.	47.0	0.0	0.0	0
25	1	5	144	1000	0	88.0	7.6	25.8	3715.2	.
26	2	5	128	1000	0	38.3	7.6	0.0	0.0	96
27	4	5	148	1000	0	.	7.6	0.0	0.0	283
28	6	5	50	1000	0	13.8	7.6	0.0	0.0	100
29	5	6	50	1000	0	.	.	18.2	910.0	.
30	11	6	50	1000	0	.	.	0.0	0.0	100
31	12	6	94	1000	0	.	.	0.0	0.0	188
32	13	6	151	1000	0	.	.	0.0	0.0	302
33	4	7	50	1000	0	.	.	0.0	0.0	70
34	8	7	114	1000	0	25.0	.	25.0	2850.0	.
35	9	7	20	1000	0	.	.	0.0	0.0	40
36	7	8	114	1000	0	.	.	0.0	0.0	228
37	4	9	0	1000	0	.	29.5	19.3	0.0	.
38	7	9	20	1000	0	.	29.5	25.0	500.0	.
39	10	9	34	1000	0	.	29.5	0.0	0.0	7
40	14	9	56	1000	0	.	29.5	0.0	0.0	112
									<b>34382.8</b>	

Figure 4.5: SAS Optimal Network (Screenshot).



Firefox

CTP Solver (by: Damian Lamp)

localhost/ctpsolver/

OPTIMAL NETWORK	Tail	Head	Cost	Capacity	Lower Bound	Flow	Capacity - Flow	Reduced Cost	Order	Basis
2147483647	1	15	1000	M	0	0.0	M	0	0	-
2147483646	2	15	1000	M	0	0.0	M	0	0	-
2147483645	15	3	1000	M	0	0.0	M	0	0	-
2147483644	15	4	1000	M	0	0.0	M	0	0	-
2147483643	15	5	1000	M	0	0.0	M	0	0	-
2147483642	6	15	1000	M	0	0.0	M	0	0	-
2147483641	7	15	1000	M	0	0	M	0	0	-
2147483640	8	15	1000	M	0	0	M	0	0	-
2147483639	15	9	1000	M	0	0.0	M	0	0	-
2147483638	15	10	1000	M	0	0.0	M	0	0	-
2147483637	15	11	1000	M	0	0.0	M	0	0	-
2147483636	15	12	1000	M	0	0.0	M	0	0	-
2147483635	15	13	1000	M	0	0.0	M	0	1	basic
2147483634	15	14	1000	M	0	0.0	M	0	0	-
OPTIMAL NETWORK	Tail	Head	Cost	Capacity	Lower Bound	Flow	Capacity - Flow	Reduced Cost	Order	Basis
1	1	2	112	M	0	62.2	937.8	0	5	basic
3	2	3	124	M	0	34.2	965.8	0	6	basic
4	2	4	167	M	0	66.3	933.7	0	7	basic
6	3	4	172	M	0	0	M	-129	0	-
2	1	5	144	M	0	25.8	974.2	0	4	basic
5	2	5	128	M	0	0.0	M	-96	0	-
7	5	4	148	M	0	0	M	-13	0	-
10	5	6	50	M	0	18.2	981.8	0	3	basic
8	7	4	50	M	0	0	M	-30	0	-
14	8	7	114	M	0	25	975	0	10	basic
9	4	9	0	M	0	19.3	980.7	0	8	basic
15	7	9	20	M	0	25	975	0	9	basic
16	10	9	34	M	0	0.0	M	-7	0	-
11	6	11	50	M	0	12.5	987.5	0	12	basic
18	11	10	8	M	0	9.0	991.0	0	13	basic
12	6	12	94	M	0	6.0	994.0	0	14	basic
13	6	13	151	M	0	13.5	986.5	0	2	basic
19	12	13	86	M	0	0	M	-29	0	-
17	9	14	56	M	0	14.8	985.2	0	11	basic
20	14	13	56	M	0	0	M	-46	0	-

Microsoft-IPS/7.0

Figure 4.6: CTP Solver Optimal Network (Screenshot, Light CSS).

### **4.3. Accuracy Summary**

Accuracy was determined by comparing the resulting optimal network flow computed by the CTP Solver to the optimal network flow computed separately by both AMPL and SAS for the same network.

The CTP Solver, AMPL, and SAS each employ optimal algorithms, so it should be expected that they all obtain the same optimal result for the test networks. This was the case with the tests performed and since many optimal solvers currently exist, new contributions should focus on improvements in the areas of performance and ease of use.

Because ease of use is inherently subjective, the software performance in terms of speed should be considered the best measure of comparison between the three solutions.

### **4.4. Performance Summary**

On smaller networks, up to and including the IEEE 118-Bus Test System, there was little difference between AMPL, SAS, or the CTP Solver. The computation time was low enough that the measurement precision could be questioned due to the way processing time is essentially estimated using the system clock for the CTP Solver.

The true performance comparison came from the larger randomly generated test networks. Unfortunately the Student License version of AMPL was only able to test up to the IEEE 118-Bus Test System so it was primarily just a contest between the CTP Solver and SAS.

Sadly, the CTP Solver was destroyed by SAS on larger networks; it wasn't even close. Somehow, and very surprisingly, both the AMPL LPSOLVE solver and the SAS "netflow" procedure were able to maintain a very consistent execution time throughout all tests, even when the size of the network increased. The expectation was for the execution time to become

progressively higher as the node and arc counts grew, as was the case with the CTP Solver.

During the implementation of the CTP Solver, the hope was that LINQ queries would be fast enough to overcome the recursive traversal through the entire basis tree for each simplex iteration. Unfortunately this likely contributed to its poor performance on larger sized networks.

While an extremely disappointing outcome, the CTP Solver is not a wasted effort by any means. Improvements can clearly be made to its modified simplex process and some of the possibilities are outlined in the conclusion. Its handling of bidirectional arcs is also an encouraging innovation that could be utilized in other systems.

If the network size were small enough, the CTP Solver might be a potential optimal self-healing method for the smart grid. But realistically, the CTP Solver is simply too slow in its current state to be considered a viable solution.

Interestingly, a couple of the CTP Solver's performance improvements had already been implemented before comparing the results to SAS.

#### **4.4.1. CTP Solver Output Performance Improvements**

For the larger test network comparisons, displaying each simplex iteration and cycle traversal was unnecessary bloat. The configuration allows each of these display options to be shown or hidden, allowing user to decide whether or not the CTP Solver should output the information.

Choosing to only display the final, optimal network provides significant speed improvements. For each of the following tests, ten runs were made for each average solve time along with the final results from the application comparison runs, providing an approximate general performance result.

#### 4.4.1.2. Hiding Simplex Iterations

The CTP Solver can display the entire network topology at every simplex iteration, allowing the user to step through the results and follow the solver's decisions. This can be beneficial when manually calculating solutions, such as verifying student results for assignments in an academic course.

However, for large networks, manual solution calculations are simply infeasible, which is the entire purpose of software such as the CTP Solver.

As such, the output for these individual simplex iterations can be set to not be displayed, greatly improving the CTP Solver's execution time performance.

Table 4.1: CTP Solver Performance Improvement - Hiding Simplex Iterations.

Average Solve Time (seconds)	Display All	Hide Simplex Iterations	Improvement
IEEE 30-Bus	0.071875	0.00625	<b>11.5x</b> <b>(1050%)</b>
IEEE 57-Bus	0.290625	0.0234375	<b>12.4x</b> <b>(1140%)</b>
IEEE 118-Bus	2.529513889	0.168402778	<b>15.02x</b> <b>(1402%)</b>

#### 4.4.1.3. Hiding Cycle Debugging Log

In the same vein as hiding the simplex iterations, the cycle debugging log can also be removed from the output display using a configuration setting.

As with the simplex iterations, it is recommended to not display the cycle debugging log for large networks as the CTP Solver's execution time is slightly reduced when not displaying this information. While not a

significant improvement as was the case when hiding the simplex iterations, every bit helps.

Table 4.2: CTP Solver Performance Improvement - Hiding Cycle Debugging Log.

Average Solve Time (seconds)	Display All	Hide Cycle Debugging Log	Improvement
IEEE 30-Bus	0.071875	0.059375	<b>1.21x (21%)</b>
IEEE 57-Bus	0.290625	0.28125	<b>1.03x (3%)</b>
IEEE 118-Bus	2.529513889	2.310763889	<b>1.09x (9%)</b>

#### 4.4.1.4. Hiding Cycle Debugging Log and Simplex Iterations

When both the simplex iterations and cycle debugging log are set to not display, the performance gains are naturally significant.

In the case of the IEEE 118-Bus test, it was noted that the improvement was slightly less than when only hiding the simplex iterations. This is likely due to the fact the cycle debugging log results are fairly trivial and the test runs for the application comparisons were conducted on a different day than the simplex and cycle debugging improvement tests.

With such small test sample sizes, the machine could have been influenced different processes running while the tests were conducted.

Despite this minor inconsistency, it should be generally obvious the approximate improvement is significant enough to warrant hiding both the simplex iterations and cycle debugging log for the best performance results.

Table 4.3: CTP Solver Performance Improvement - Hiding Cycle Debugging Log and Simplex Iterations.

Average Solve Time (seconds)	Display All	Hide All	Improvement
IEEE 30-Bus	0.071875	0.00625	11.5x (1050%)
IEEE 57-Bus	0.290625	0.021875	13.29x (1229%)
IEEE 118-Bus	2.529513889	0.16875	14.99x (1399%)

#### 4.4.1.5. First Reduced Cost Arc

Another attempt at improving the CTP Solver's performance was altering the pricing or reduced cost calculations (Step 2). In order to choose the best candidate arc to enter the basis, this process iterates through every non-basic arc.

While the calculations are fast on current machines, the number of comparisons can potentially be decreased by three orders of magnitude in networks with thousands of arcs.

Unfortunately, however, choosing the first attractive arc also has a tendency to require more iterations of the entire simplex process. Likely due to the CTP Solver's traversal of the entire basis tree for each simplex iteration, it was actually detrimental to the overall performance when the first candidate arc heuristic was used instead of fully calculating the optimal arc each iteration.

### **4.5. Testing Environment and Setup**

In order to maintain consistency for each of the three applications being compared, the test networks were solved on the same machine.

#### **4.5.1. Hardware and Software**

All tests were performed on an Intel Core 2 Quad 2.67GHz processor with 4GB DDR2 800 RAM, running on Windows Vista x64. Both AMPL and SAS are standalone software applications but the CTP Solver requires a web server so a local virtual directory was created for it using IIS, running the .NET 4.0 framework.

#### **4.5.2. Test Network Setup**

Each of the IEEE test systems were read by the network generator to obtain their respective topologies. The actual distances between nodes were utilized for the IEEE 14-Bus System arc costs whereas all other test networks used randomly generated cost values.

The Custom 400- and 500-Node test systems were completely generated by the network generator application with a few changes to the final topologies in order to ensure feasibility.

#### **4.6. Test Network Results**

All three software applications were compared using the IEEE 14-Bus Test System, IEEE 30-Bus Test System, IEEE 57-Bus Test System, and IEEE 118-Bus Test System. From there, the Student License version of AMPL was unable to calculate the results due to variable and constraint limits, so only the CTP Solver and SAS were used in comparing the IEEE 300-Bus System and the Custom 400- and 500-Node Systems.

The "LPSOLVE" solver was used in AMPL, and the "netflow" procedure was used in SAS. As mentioned previously, the AMPL solver implements a modified simplex algorithm while the SAS procedure uses an interior point method. The CTP Solver was run with the simplex iterations and cycle debugging log options turned off.

Since the CTP Solver is a web application, it was tested in Firefox 19.0.2. Despite the fact that each run of a given test file is a separate HTML POST request and will be executed on demand, the browser was restarted before each test to ensure no instance caching occurred, which would create an unfair advantage.

In order to keep the tests as similar as possible, the "reset" command was given to AMPL in order to provide a fresh solution environment, and the SAS application was closed and restarted before running a given test.

During initial tests, an unexpected discovery was made about the SAS software. The application was originally not restarted before running successive tests and its results were noticeably better after its initial execution.

The results were then recalculated using a fresh instance of the SAS software for each test since it was clear the application was in some way storing information from the previous tests to speed up future runs. This made the SAS results much more consistent and thus more indicative of its true execution time performance.

The "\_total\_solve\_time" value was used in determining the execution time for AMPL, the "cpu time" of only the "netflow" procedure was used for SAS, and the "Solver Execution Time" was used for the CTP Solver results.

Using the "real time" value in SAS for the "netflow" procedure might have been a closer representation of the CTP Solver's calculation since it simply uses the elapsed time of the system clock, but its results would have been fairly similar as far as the overall average results were concerned.

For each network, a series of ten consecutive runs were executed for each solver and an average of these runs was taken as the solver's general performance time. While this is admittedly a small sample size, the intent was to simply make a pedestrian comparison between the three solutions.

#### **4.6.1. IEEE 14-Bus Results: Distance Cost**

Overall there is no discernible difference between any of the solvers from a user's perspective in terms of execution speed. The computations are essentially instantaneous in all three applications since it is such a small network.



The CTP Solver had the best performance for this network but in a larger sample size, AMPL could very well have done better.

#### 4.6.1.1. Accuracy

All three applications reached the same objective function value of 34,382.8 in every test. SAS used 21 iterations while AMPL and the CTP Solver each needed only 15.

#### 4.6.1.2. Performance

The CTP Solver had the best time, followed by AMPL and then SAS. For some reason, AMPL required two solve statement executions, claiming the supply and demand values were not equal in the first run and thus failing to execute before calculating the solution with the second solve command.

An attempt was made to ensure all supply and demand values had the same number of significant digits after the decimal, but the result was the same. Since it accurately determines the optimal solution with the second solve statement (despite identical model and data files), this behavior was dismissed as a quirk of the LPSOLVE solver.

Table 4.4: IEEE 14- Bus Performance Results.

Solve Time (seconds)	AMPL (15 iterations)	SAS (21 iterations)	CTP Solver (16 iterations)
Test 1	0.015625	0.18	0.015625
Test 2	0.015625	0.21	0
Test 3	0	0.23	0
Test 4	0	0.25	0
Test 5	0	0.2	0
Test 6	0.015625	0.23	0
Test 7	0	0.21	0

Table 4.4: IEEE 14-Bus Performance Results (continued).

Solve Time (seconds)	AMPL (15 iterations)	SAS (21 iterations)	CTP Solver (16 iterations)
Test 8	0	0.23	0
Test 9	0.015625	0.18	0
Test 10	0	0.18	0
Average	0.00625	0.21	<b>0.0015625</b>

#### 4.6.1.3. Performance Graphs

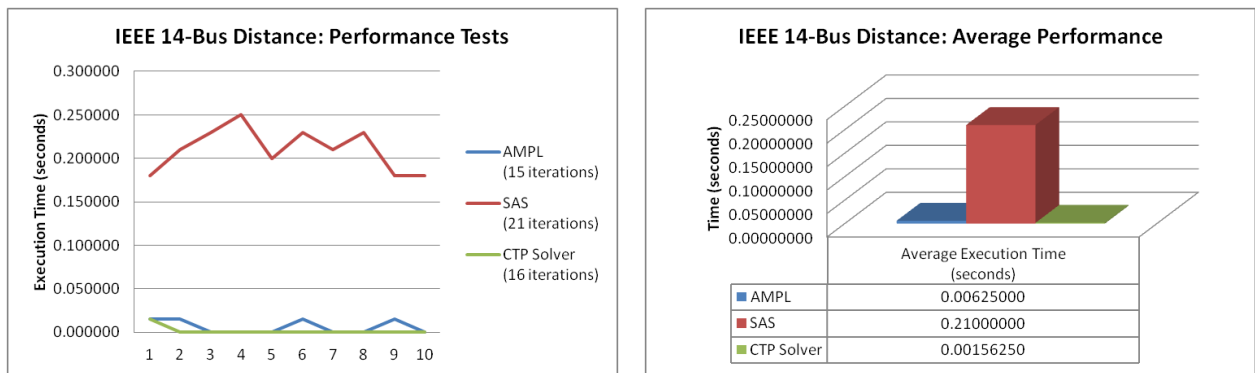


Figure 4.7: IEEE 14-Bus Performance Graphs.

#### 4.6.2. IEEE 30-Bus Results: Random Cost

The CTP Solver came out on top again in this small network, but just as with the 14-Bus System, AMPL could have had a better overall performance with a larger test sample size.

##### 4.6.2.1. Accuracy

All three applications again arrived at the same optimal solution value 23,692.6396. The CTP Solver again had the fewest iterations at 31, with AMPL and SAS following, requiring 35 and 37 iterations, respectively.

#### 4.6.2.2. Performance

Table 4.5: IEEE 30-Bus Performance Results.

Solve Time (seconds)	AMPL (35 iterations)	SAS (37 iterations)	CTP Solver (31 iterations)
Test 1	0.015625	0.23	0.03125
Test 2	0	0.24	0.015625
Test 3	0.015625	0.25	0
Test 4	0	0.21	0
Test 5	0	0.26	0.015625
Test 6	0.015625	0.2	0
Test 7	0.015625	0.23	0
Test 8	0.015625	0.23	0
Test 9	0.015625	0.21	0
Test 10	0	0.18	0
Average	0.009375	0.224	<b>0.00625</b>

#### 4.6.2.3. Performance Graphs

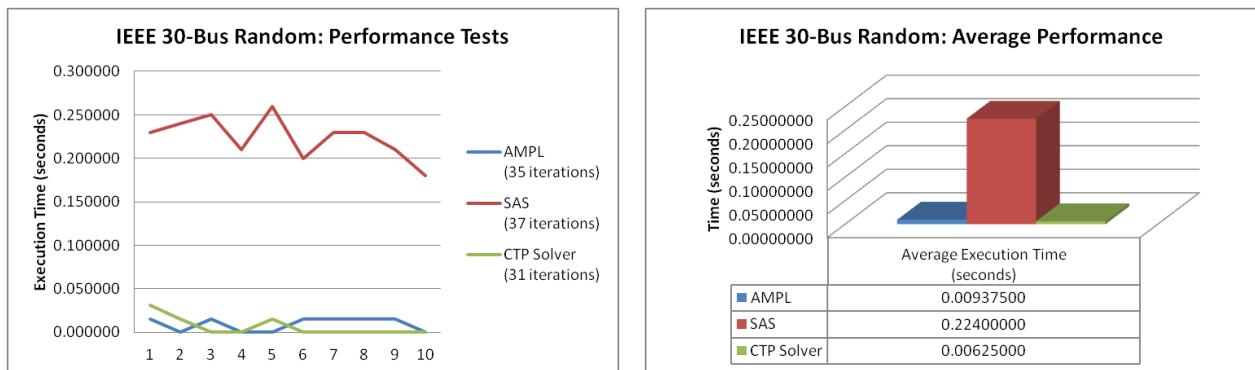


Figure 4.8: IEEE 30-Bus Performance Graphs.

### 4.6.3. IEEE 57-Bus Results: Random Cost

AMPL chalked up its first win in this network with results calculated more than two times faster than the CTP Solver. However, the small network size means the difference in results calculations are still essentially unnoticeable from the user perspective.

Despite AMPL being the new speed winner, the iteration trend continued with the same order.

#### 4.6.3.1. Accuracy

Again all three applications determined the optimal solution to be 35508.5215. As with the previous two networks, the CTP Solver had the fewest iterations (63) with AMPL (80) and SAS (83) following in the same order.

#### 4.6.3.2. Performance

AMPL had the best time with the CTP Solver and SAS following. It is interesting to note that both AMPL and SAS had faster average times for the 57-Bus System than for the 30-Bus System. Even with such small network sizes, it should be expected for the applications to increase at least slightly as the network sizes grow larger.

An odd pattern emerged while running all but the final test between AMPL and the CTP Solver; each appeared to follow the same sequence of two values 0.015625 seconds apart. AMPL alternated irregularly between 0 and 0.015625 while the CTP Solver alternated between 0.015625 and 0.03125. The respective patterns between the larger and smaller values were identical until the final test.

Table 4.6: IEEE 57-Bus Performance Results.

Solve Time (seconds)	AMPL (80 iterations)	SAS (83 iterations)	CTP Solver (63 iterations)
Test 1	0.015625	0.23	0.03125
Test 2	0.015625	0.25	0.03125
Test 3	0	0.2	0.015625
Test 4	0	0.2	0.015625
Test 5	0.015625	0.25	0.03125
Test 6	0	0.21	0.015625
Test 7	0.015625	0.18	0.03125
Test 8	0	0.2	0.015625
Test 9	0	0.2	0.015625
Test 10	0.015625	0.2	0.015625
Average	<b>0.0078125</b>	0.212	0.021875

4.6.3.3. Performance Graphs

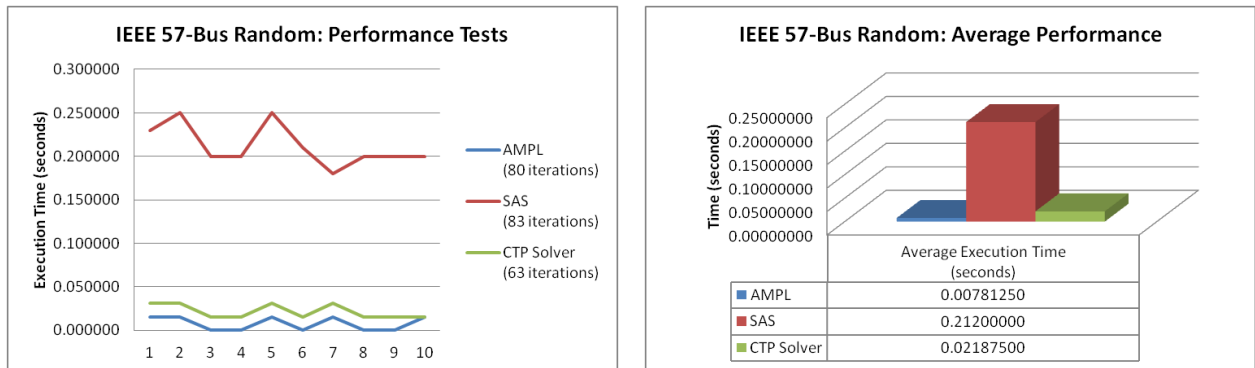


Figure 4.9: IEEE 57-Bus Performance Graphs.

**4.6.4. IEEE 118-Bus Results: Random Cost**

In its last test, AMPL is victorious with the CTP Solver and SAS respectively following. The CTP Solver is starting to distance itself relatively significantly in terms of iterations.

#### 4.6.4.1. Accuracy

All three applications arrived at an optimal solution of 561,209.7657. SAS required 226 iterations, AMPL required 197 iterations, and the CTP Solver only required 148 iterations.

#### 4.6.4.2. Performance

Once again AMPL was the fastest and had the same performance as its 30-Bus System test. SAS also remained consistent, albeit as the slowest application. The CTP Solver was in the middle and is the only application with a steady increase in execution time in relation to the network size.

Table 4.7: IEEE 118-Bus Performance Results.

Solve Time (seconds)	AMPL (197 iterations)	SAS (226 iterations)	CTP Solver (148 iterations)
Test 1	0.015625	0.28	0.171875
Test 2	0.015625	0.21	0.171875
Test 3	0.015625	0.2	0.171875
Test 4	0	0.2	0.171875
Test 5	0	0.26	0.171875
Test 6	0.015625	0.25	0.171875
Test 7	0	0.25	0.15625
Test 8	0.015625	0.21	0.171875
Test 9	0	0.18	0.171875
Test 10	0.015625	0.18	0.15625
Average	<b>0.009375</b>	0.222	0.16875

#### 4.6.4.3. Performance Graphs

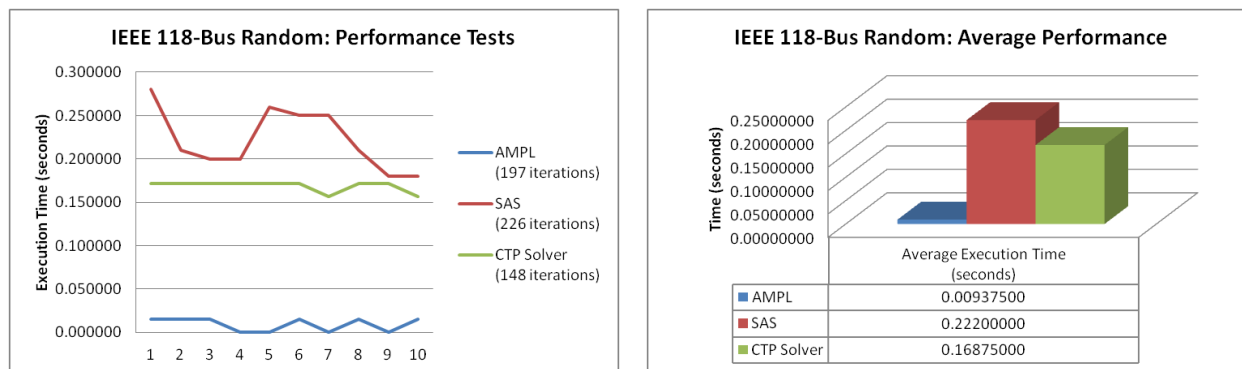


Figure 4.10: IEEE 118-Bus Performance Graphs.

#### 4.6.5. IEEE 300-Bus Results: Random Cost

This was the first test excluding AMPL and also the first network where the results started indicating separation between applications. SAS was the clear winner and the difference between its computation time and the CTP Solver would for the first time be noticeable.

##### 4.6.5.1. Accuracy

Both SAS and the CTP Solver reached an optimal value of 6,348,472.507. The CTP Solver only required 392 iterations whereas SAS required 629.

##### 4.6.5.2. Performance

Despite needing fewer iterations, the CTP Solver was approximately an order of magnitude slower than SAS. The CTP Solver is starting a troubling trend toward exponential growth in its execution time.

Table 4.8: IEEE 300-Bus Performance Results.

Solve Time (seconds)	SAS (629 iterations)	CTP Solver (392 simplex iterations)
Test 1	0.21	2.28125

Table 4.8: IEEE 300-Bus Performance Results (continued).

Solve Time (seconds)	SAS (629 iterations)	CTP Solver (392 simplex iterations)
Test 2	0.22	2.265625
Test 3	0.24	2.28125
Test 4	0.26	2.28125
Test 5	0.2	2.28125
Test 6	0.21	2.28125
Test 7	0.23	2.265625
Test 8	0.23	2.265625
Test 9	0.23	2.25
Test 10	0.2	2.28125
Average	<b>0.223</b>	2.2734375

#### 4.6.5.3. Performance Graphs

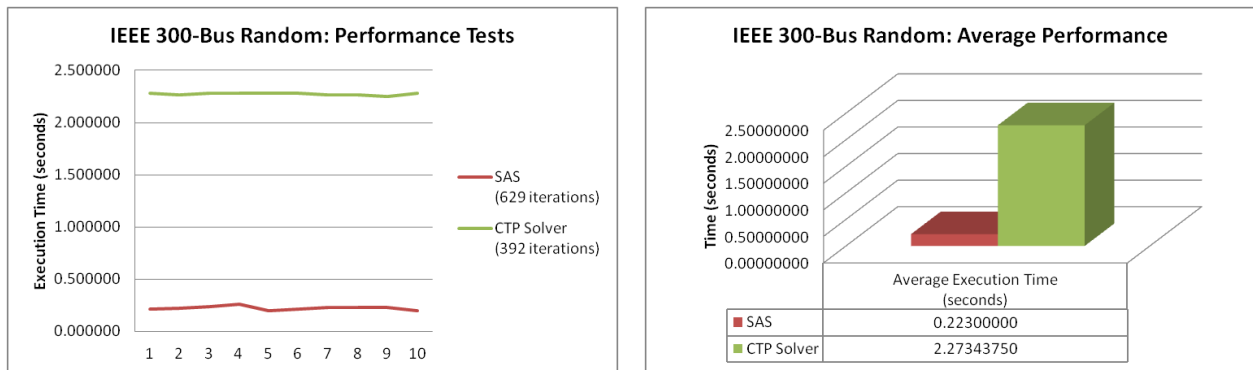


Figure 4.11: IEEE 300-Bus Performance Graphs.

#### 4.6.6. Custom 400 Node Results: Random Cost

The results for this network weren't even close in terms of performance as SAS was significantly faster than the CTP Solver. This was certainly an unexpected result and one that will hopefully be corrected with some modifications to the CTP Solver.



A silver lining is that the CTP Solver takes nearly half as many iterations as SAS to calculate the optimal result.

#### 4.6.6.1. Accuracy

The CTP Solver and SAS both had the same optimal solution of 192,195.0485. SAS continues to require many more iterations than the CTP Solver (942 compared to 521).

#### 4.6.6.2. Performance

The exponential growth trend continues for the CTP Solver while SAS remains relatively steady in its calculation times.

Table 4.9: Custom 400 Node Performance Results.

Solve Time (seconds)	SAS (942 iterations)	CTP Solver (521 iterations)
Test 1	0.2	9.75
Test 2	0.18	9.546875
Test 3	0.25	9.75
Test 4	0.23	9.8125
Test 5	0.25	10
Test 6	0.28	9.53125
Test 7	0.2	9.609375
Test 8	0.25	9.859375
Test 9	0.21	9.765625
Test 10	0.26	9.8125
Average	<b>0.231</b>	9.74375

#### 4.6.6.3. Performance Graphs

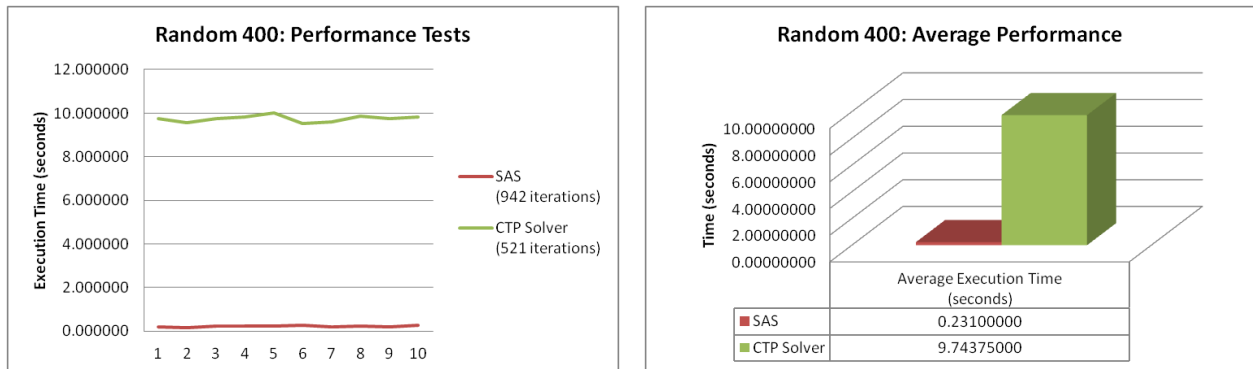


Figure 4.12: Custom 400 Node Performance Graphs.

#### 4.6.7. Custom 500 Node Results: Random Cost

The final test between the CTP Solver and SAS brought about more of the same results with SAS maintaining very fast times while the CTP Solver continued its trend toward exponential growth.

##### 4.6.7.1. Accuracy

Both systems calculated an optimal solution of 2,485,906.7415. SAS was able to maintain its significant speed advantage while taking more than twice as many iterations as the CTP Solver (1562 for SAS compared to 708 for the CTP Solver).

##### 4.6.7.2. Performance

Once again SAS remained largely consistent in its execution time while the CTP Solver took much, much longer. This test shows the CTP Solver would certainly not be fast enough in its current state to be a viable smart grid optimization solution.

Table 4.10: Custom 500 Node Performance Results.

Solve Time (seconds)	SAS (1286 iterations)	CTP Solver (708 iterations)
Test 1	0.31	30.125
Test 2	0.28	27.828125
Test 3	0.22	28.125
Test 4	0.23	28.21875
Test 5	0.23	28.140625
Test 6	0.26	29.5
Test 7	0.26	27.96875
Test 8	0.28	28.171875
Test 9	0.24	28.203125
Test 10	0.17	28.09375
Average	<b>0.248</b>	28.4375

4.6.7.3. Performance Graphs

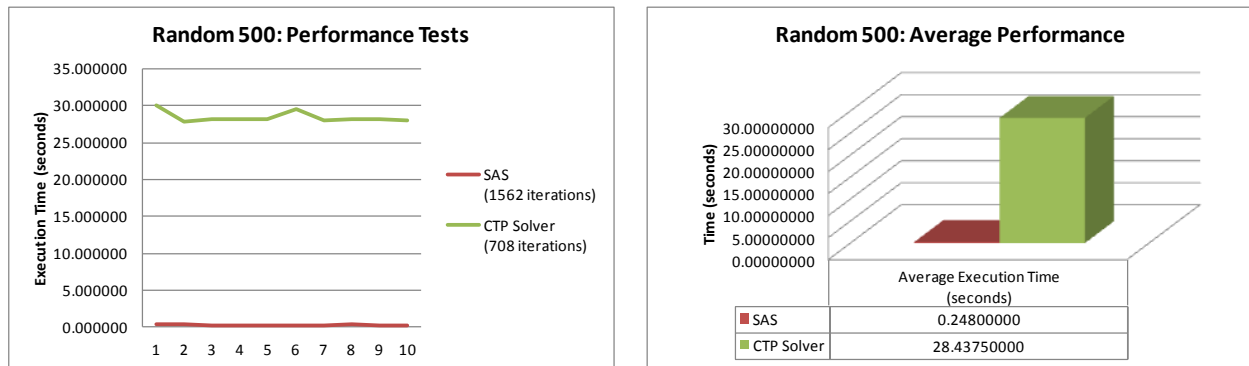


Figure 4.13: Custom 500 Node Performance Graphs.

**4.7. Results Analysis**

In small network tests, the CTP Solver excelled over SAS and also had slightly better performance than AMPL. However, the larger the networks became, the worse the CTP Solver performed.

SAS was able to keep its growth pattern fairly linear as networks with more nodes and arcs were calculated. Unfortunately the CTP Solver's growth pattern appears to be exponential. This is likely due to its traversal of the entire basis tree for each simplex iteration, making the time complexity approximately  $O(n^2)$  where "n" is the number of nodes in the network.

It was hoped that the use of LINQ queries would compensate, but clearly the way the algorithms were implemented, that wasn't the case, proving math trumps faith.

An ongoing effort is attempting to resolve this issue while staying true to the use of LINQ queries but it might come down to reverting to traditional linear programming techniques as the best approach.

The following Figure 4.14 shows the average performance results starting from the IEEE 14-Bus System and progressing through to the Random 500-Node test network. Only SAS and the CTP Solver were graphed due to the fact AMPL was only tested on the four smallest networks.

The graph clearly shows the exponential growth pattern of the CTP Solver and the linear nature of SAS.

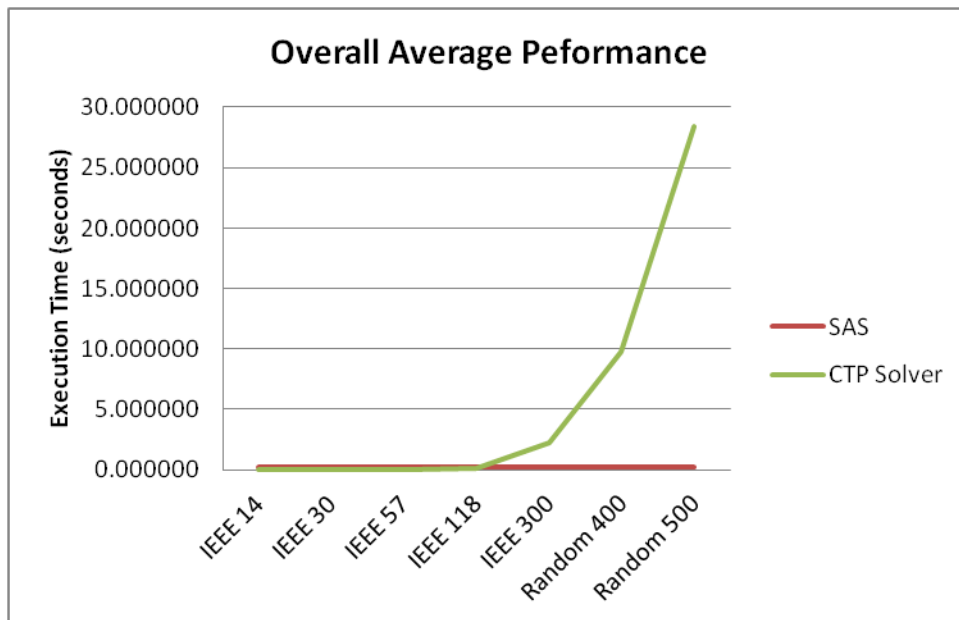


Figure 4.14: Overall Average Performance.

One silver lining in the CTP Solver's performance was its consistency in requiring the fewest iterations to calculate its results. This leads to the possibility of it potentially being much, much faster in a parallelized implementation.

## 5. CONCLUSION

In an attempt to address the problem of smart grid optimization based on various performance measures potentially related to self-healing solutions, a custom capacitated transshipment problem solver (or CTP Solver) was developed using ASP.NET C#.

Through custom objects and an algorithmic process based on the simplex method, the CTP Solver calculates an optimal solution to a supplied network file or database and displays the results to the user while also exporting the optimal network to XML, spreadsheet, and updates the database if used as the original network source.

The CTP Solver contributes the application of object oriented software development concepts to the solution of linear programming network flow problems as well as an innovation in the process of handling bidirectional arcs. It is also an universally accessible web application as opposed to traditional standalone software.

However, in its current state, the CTP Solver would not be a viable solution for smart grid optimization due to its exponentially slow performance. A self-healing system for the smart grid would need to be as fast as possible to help prevent cascading failures; the CTP Solver is currently just not fast enough at solving large network problems, but could be viable if the solution space was limited to smaller networks.

Even though its speed performance compared to SAS was extremely disappointing on larger-scale problems, the CTP Solver is still a valuable tool with plenty of room for improvements. With a better process for basis updates that still takes advantage of LINQ queries, the CTP Solver might be capable of reaching performance results similar to SAS and become a crucial solution to the problem of smart grid self-healing among other network optimization applications.

## **5.1. Primary Contributions**

The CTP Solver's primary contributions can be summarized in three main areas:

1. Architecture and Platform
2. User Experience
3. Bidirectional Arcs Algorithm Innovation

### **5.1.1. Architecture and Platform**

The architecture and platform contributions of the CTP Solver include:

1. Web Application
2. Object-Oriented Architecture and Concepts
3. LINQ

#### **5.1.1.1. Web Application**

Many of the current solutions in the category of optimization applications are programs that must be installed on a user's machine. This inherently adds a layer of complexity to the requirements for using the application. Sometimes, as in the case of SAS, the installation process can arguably be more time intensive than determining how to actually use the software.

The CTP Solver breaks this traditional software mold, allowing users to easily access the solution results of their problems without installing any software.

As a web application, the CTP Solver is available to any device with an internet connection. Since the solution processing is done on the server, the user device does not need to be powerful enough to make the calculations.

#### 5.1.1.2. Object-Oriented Architecture and Concepts

By implementing an object-oriented architecture, updates and changes to the CTP Solver's code are more easily maintained. This allows developers to quickly determine the areas of the application that need updates and make the changes in a timely manner.

Using object-oriented design also provides developers with a familiar structure and easy-to-use code environment. The CTP Solver's code reads logically with documented methods explaining the processes and reasons behind the design choices.

#### 5.1.1.3. LINQ

One of the primary unknowns of the CTP Solver's design was whether or not LINQ would be a beneficial feature to implement.

Since LINQ queries allow coded data structures to function similarly to a database, using LINQ seemed to be a good choice for handling interactions with the CTP Solver's data.

While LINQ certainly made writing the code more clear and easy to follow, the current implementation might not be the best approach in terms of execution speed. However, with some modifications to the code, it is still hoped that LINQ can be implemented with acceptable performance. This would make the CTP Solver a viable solution for smart grid optimization.

#### **5.1.2. User Experience**

The user experience contributions of the CTP Solver include:

1. Ease of Use
2. Standardized Data Format
3. No New Languages



#### 5.1.2.1. Ease of Use

It was an important design goal for the CTP Solver to be generally intuitive for its users to work with and it appears to have reached this goal.

There is very little work required of the user; the most demanding expectation is formatting the XML file correctly so the CTP Solver can properly read and calculate the results.

Plenty of information is then displayed to the user in easy-to-read tables with the added benefit of also being automatically exported to both XML and .CSV.

#### 5.1.2.2. Standardized Data Format

The main benefit of using XML for network structures and data is that it is a mature, standard, and widely-used format for transferring and manipulating data.

As such, XML is almost ubiquitously supported by software and web protocols, allowing users to utilize their data in multiple applications without having to alter the information.

#### 5.1.2.3. No New Languages

Since the CTP Solver automates so much of the processes involved with calculating results, the user does not need to learn a new application-specific language or syntax.

In the case of AMPL and SAS, the user must learn how to model their own problem solvers or search through documentation to find the correct methods and syntax required to use the application. This makes the user's learning curve much steeper for these programs than for the CTP Solver.

The user simply needs to learn how to format the network XML file and the CTP Solver does the rest.

### **5.1.3. Bidirectional Arcs Algorithm Innovation**

Probably the most exciting innovation the CTP Solver provides is a different way of handling bidirectional arcs.

Since a capacitated transshipment problem requires directed arcs, traditionally all arcs are considered separately. This means an arc between two nodes would normally require two definitions in the network data structure.

The CTP Solver simply adds a Boolean attribute to every arc, allowing it to be bidirectional. When it is bidirectional, flow is allowed in both directions between a node pair.

By handling bidirectional arcs in this manner, the CTP Solver can effectively halve the network file size and memory requirements of a dataset containing all bidirectional arcs.

## **5.2. Future Work and Improvements**

Since the CTP Solver is not currently a viable solution for smart grid self-healing, there are a number of improvements it can likely benefit from in future modifications to aspects such as parallelization, visualization, and optimization among others.

### **5.2.1. Parallelization**

One of the areas that could be parallelized is individual reduced cost arc calculations. These could be divided between the available processors for faster processing and the best reduced cost from each processor thread could be compared to find the overall best reduced cost.

Another part of the CTP Solver that might be a good candidate for parallelization is the cycle traversal. The algorithm follows the back path of both the head and tail node at the same time, starting from the deepest node. Once the two paths are at the same node depth, the maximum flow change, or theta, value could be calculated separately on the head and tail back paths.

The display of the output tables for each simplex iteration could also be parallelized between available processors, with the cycle and basis arcs being combined into one processor thread if there are only two processors. Note: this possibility should not be confused with providing separate processors different simplex iterations. Each iteration is dependent on the previous so they must be completed sequentially.

Of course, the simplex iterations can already be hidden by setting the proper attribute in the configuration settings, greatly reducing the overall execution time, so parallelization of this part of the CTP Solver might not be worth the effort.

### **5.2.2. Visualization**

While the optimal network table accurately depicts the resulting topology, additionally allowing the user to create a visual representation of the network structures would be ideal.

An interface for manipulating the graphical layout would enable the user to see an accurate representation of the network, possibly allowing further insights about the resulting information.

### **5.2.3. Optimization**

The process of updating the basis and network is currently implemented using a recursive method. It is also stepping through the entire basis tree as opposed to simply updating the arcs connecting the cycle nodes. By

finding a way to change the algorithm to only update the cycle nodes and arcs, significant performance gains are likely possible for large-scale networks. However, that might require the use of more traditional linear programming development techniques in place of LINQ queries.

## REFERENCES

- [1] Litos Strategic Communication, "The Smart Grid: An Introduction," [Online]. Available: <http://energy.gov/oe/downloads/smart-grid-introduction-0>.
- [2] U.S. Department of Energy, "What is the Smart Grid?," [Online]. Available: [www.smartgrid.gov/the\\_smart\\_grid](http://www.smartgrid.gov/the_smart_grid).
- [3] United States Department of Energy, "Smart Grid," [Online]. Available: <http://energy.gov/oe/technology-development/smart-grid>.
- [4] Institute of Electrical and Electronics Engineers, "About IEEE," [Online]. Available: <http://www.ieee.org/about/index.html>.
- [5] University of Washington, "Resources: Power Systems Test Case Archive," [Online]. Available: <http://www.ee.washington.edu/research/pstca/>.
- [6] T. M. C. James P. Ignizio, *Linear Programming*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1994.
- [7] G. Dantzig, "Activity Analysis of Production and Allocation: Proceedings of a Conference," *Cowles Commission for Research in Economics*, 1951.
- [8] N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming," *Combinatorica*, vol. 4, no. 4, pp. 373-395, 1984.
- [9] J. J. J. H. D. S. Mokhtar S. Bazaraa, *Linear Programming and Network Flows*, Hoboken, New Jersey: John Willy & Sons, Inc., 2010.
- [10] G. G. B. G. W. G. Gordon H. Bradley, "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, vol. 24, no. 1, pp. 1-34, 1977.
- [11] K. N. Prakash Ranganathan, "A LP Based Large Scale Decomposition and Optimization in Smart Grid".
- [12] Microsoft, "LINQ (Language-Integrated Query)," [Online]. Available: <http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx>.
- [13] S. S. S. R. Ellis Horowitz, *Computer Algorithms*, Summit, New Jersey: Silicon Press, 2008.
- [14] D. Knuth, *The Art of Computer Programming*, vol. 1: Fundamental Algorithms, Westford, Massachusetts: Pearson Education, Inc., 1997.
- [15] E. Dijkstra, "A Note on Two Problems in Connexion with Graphs,"

*Numerische Mathematik*, vol. 1, pp. 269-270, 1959.

- [16] I. Sommerville, *Software Engineering*, Boston, Massachusetts: Pearson Education, Inc., 2011.
- [17] P. N. Stuart Russell, *Artificial Intelligence: A Modern Approach*, Upper Saddle River, New Jersey: Pearson Education, Inc., 2010.
- [18] A. S. R. J. T. Zhifang Wang, "Generating Statistically Correct Random Topologies for Testing Smart Grid Communication and Control Networks," *Smart Grid, IEEE Transactions on*, vol. 1, no. 1, pp. 28-39, June 2010.
- [19] AMPL, "AMPL: A Modeling Language for Mathematical Programming," [Online]. Available: <http://www.ampl.com/>.
- [20] SAS, "About SAS," [Online]. Available: <http://www.sas.com/company/about/index.html>.
- [21] SAS, "PROC NETFLOW Statement," [Online]. Available: [http://support.sas.com/documentation/cdl/en/ormpug/63352/HTML/default/ormpug\\_netflow\\_sect016.htm](http://support.sas.com/documentation/cdl/en/ormpug/63352/HTML/default/ormpug_netflow_sect016.htm).
- [22] SAS, "SAS/OR(R) 9.2 User's Guide: Mathematical Programming," [Online]. Available: [http://support.sas.com/documentation/cdl/en/ormpug/59679/HTML/default/viewer.htm#netflow\\_sect63.htm](http://support.sas.com/documentation/cdl/en/ormpug/59679/HTML/default/viewer.htm#netflow_sect63.htm).

## APPENDIX

### **A.1. IEEE 14-Bus System Example**

The following is a step-by-step pictorial example of the CTP Solver's iterative process through the IEEE 14-Bus test system with distance as a cost measure.

In each step after the initial Step 0, the iteration's cycle flow updates are shown in the left diagram and the basis tree updates are shown in the right diagram. Then the iteration's resulting network arc flows are shown in the left diagram and the basis tree is shown in the right diagram.

Nodes are numbered and circled, artificial arcs are blue, real arcs are black, arc flows are boxed, reflected arcs are denoted in the basis tree with an asterisk, cycle arcs are bold, cycle nodes are filled black, the cycle direction is shown in orange, the entering arc is red, decremented arc flows are filled pink, and incremented arc flows are filled green.

#### **A.1.1. IEEE 14-Bus System: Step 0**

Step 0 is the original network provided to the CTP Solver. Recall from the initialization step of the CTP Solver's process that the initial basis is created using artificial arcs with flow equal to the supply or demand of the real node.

The purpose of this step is to give the CTP Solver an initial feasible basis to start its iterations.

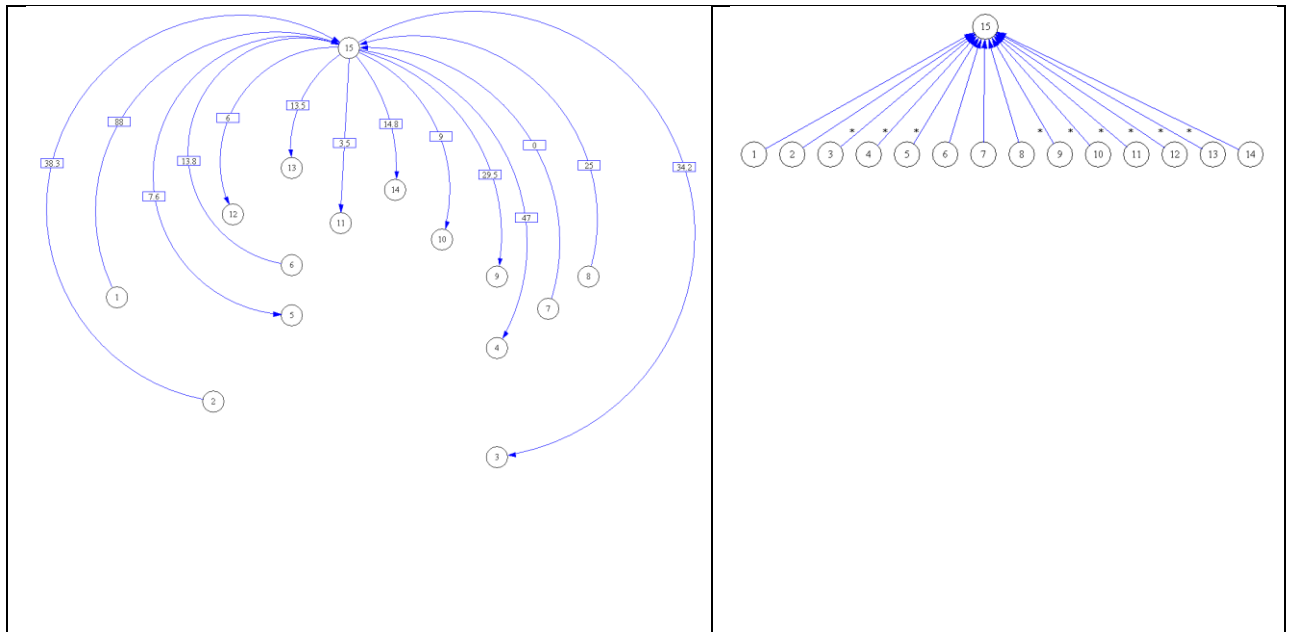


Figure A.1: IEEE 14-Bus System Step 0.

**A.1.2. IEEE 14-Bus System: Step 1 Cycle**

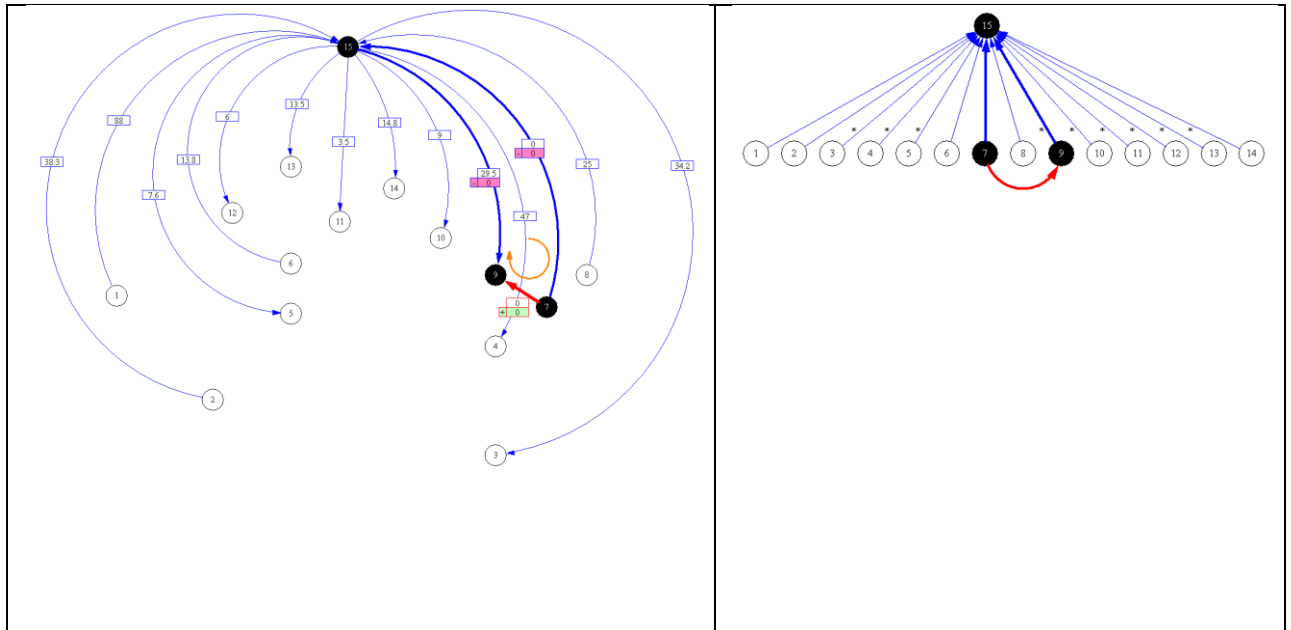


Figure A.2: IEEE 14-Bus System Step 1 Cycle.



**A.1.3. IEEE 14-Bus System: Step 1 Flows**

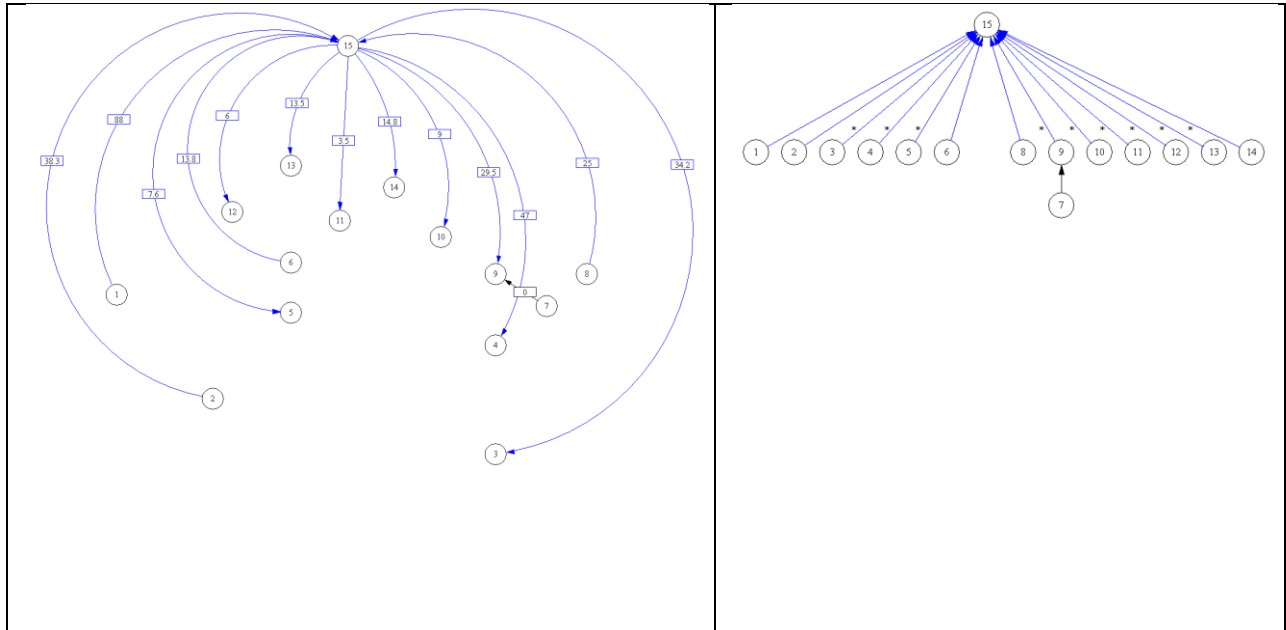


Figure A.3: IEEE 14-Bus System Step 1 Flows.

**A.1.4. IEEE 14-Bus System: Step 2 Cycle**

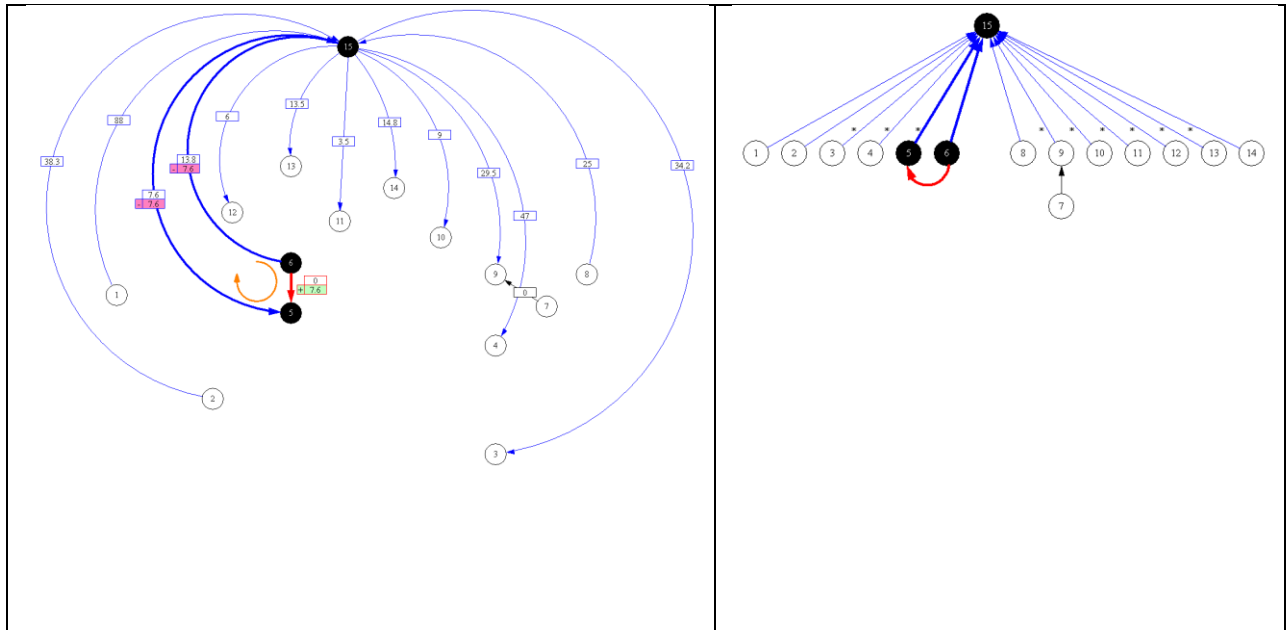


Figure A.4: IEEE 14-Bus System Step 2 Cycle.

**A.1.5. IEEE 14-Bus System: Step 2 Flows**

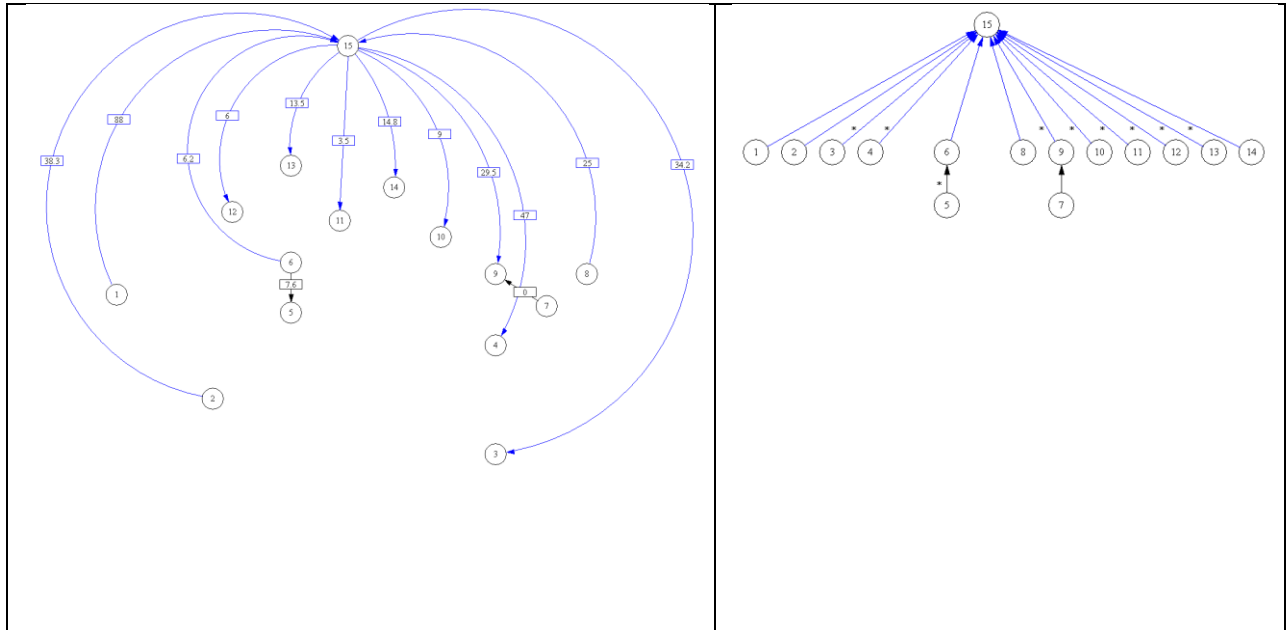


Figure A.5: IEEE 14-Bus System Step 2 Flows.

**A.1.6. IEEE 14-Bus System: Step 3 Cycle**

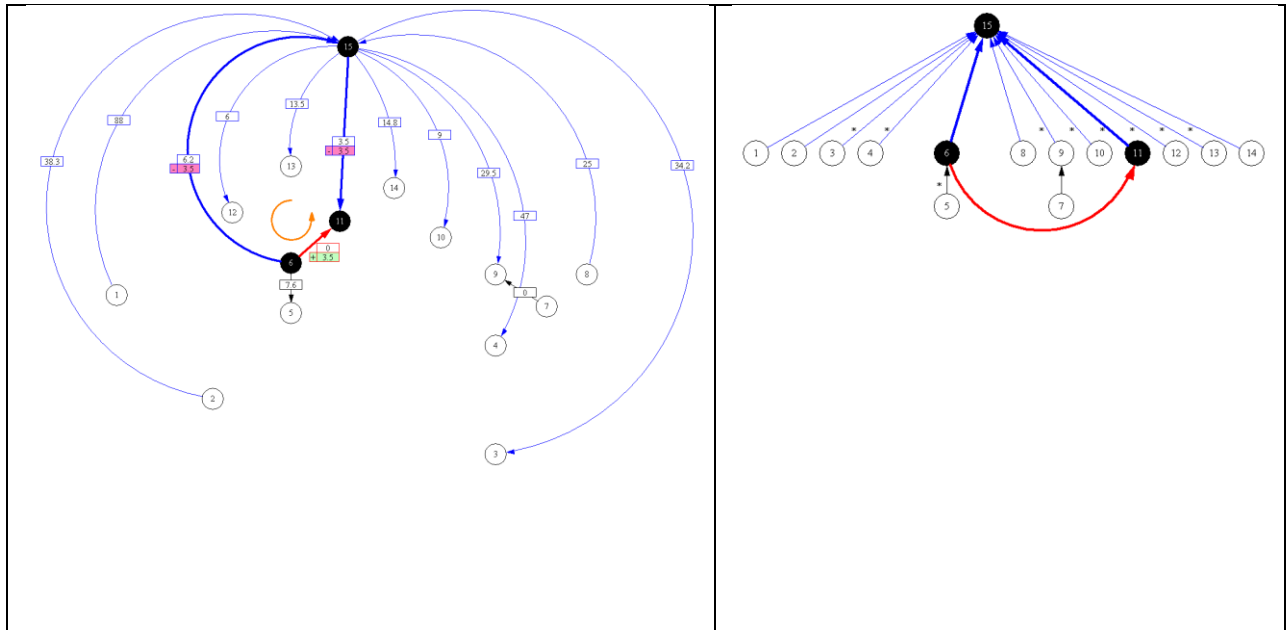


Figure A.6: IEEE 14-Bus System Step 3 Cycle.

**A.1.7. IEEE 14-Bus System: Step 3 Flows**

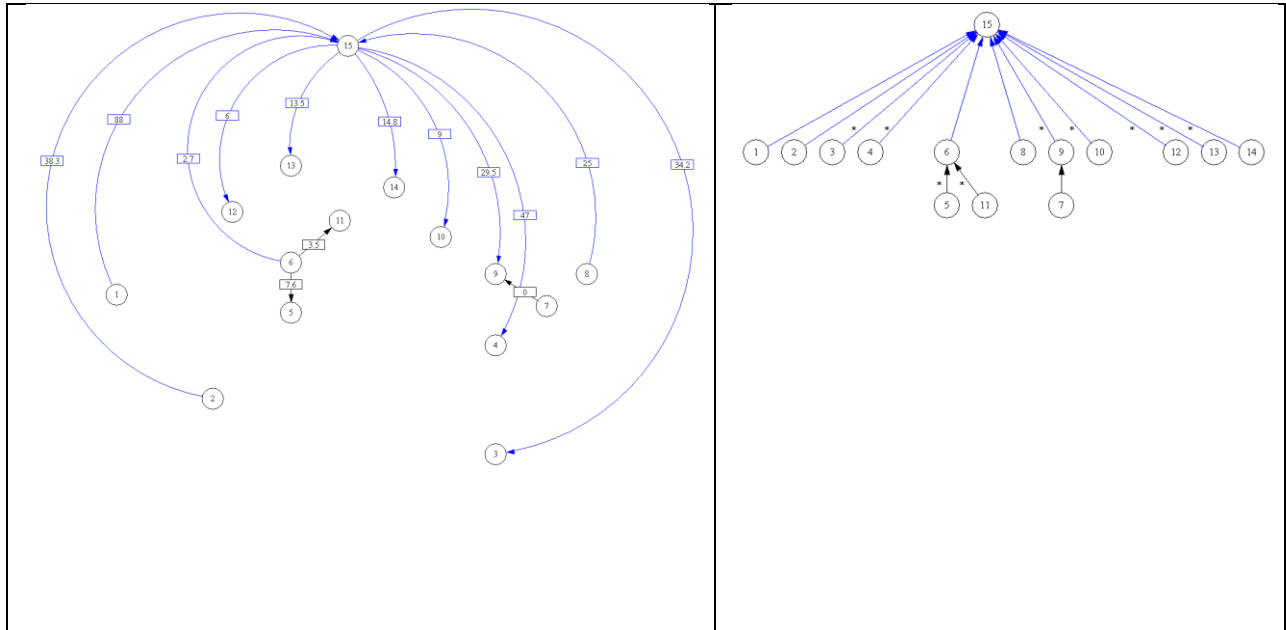


Figure A.7: IEEE 14-Bus System Step 3 Flows.

**A.1.8. IEEE 14-Bus System: Step 4 Cycle**

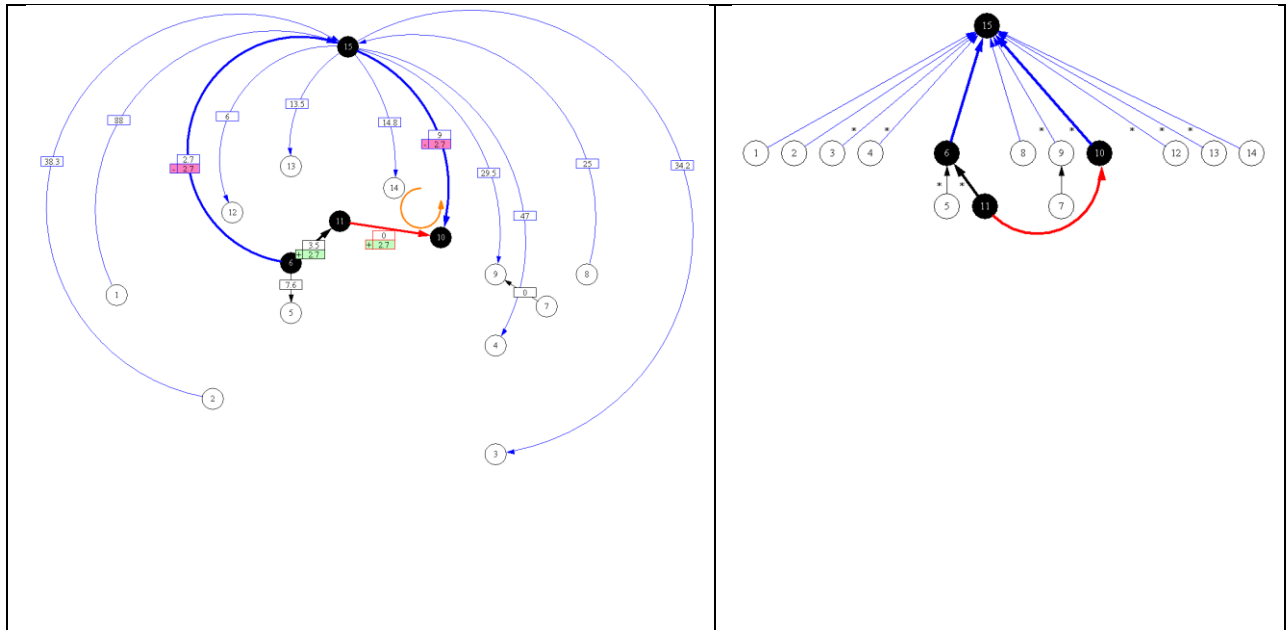


Figure A.8: IEEE 14-Bus System Step 4 Cycle.

**A.1.9. IEEE 14-Bus System: Step 4 Flows**

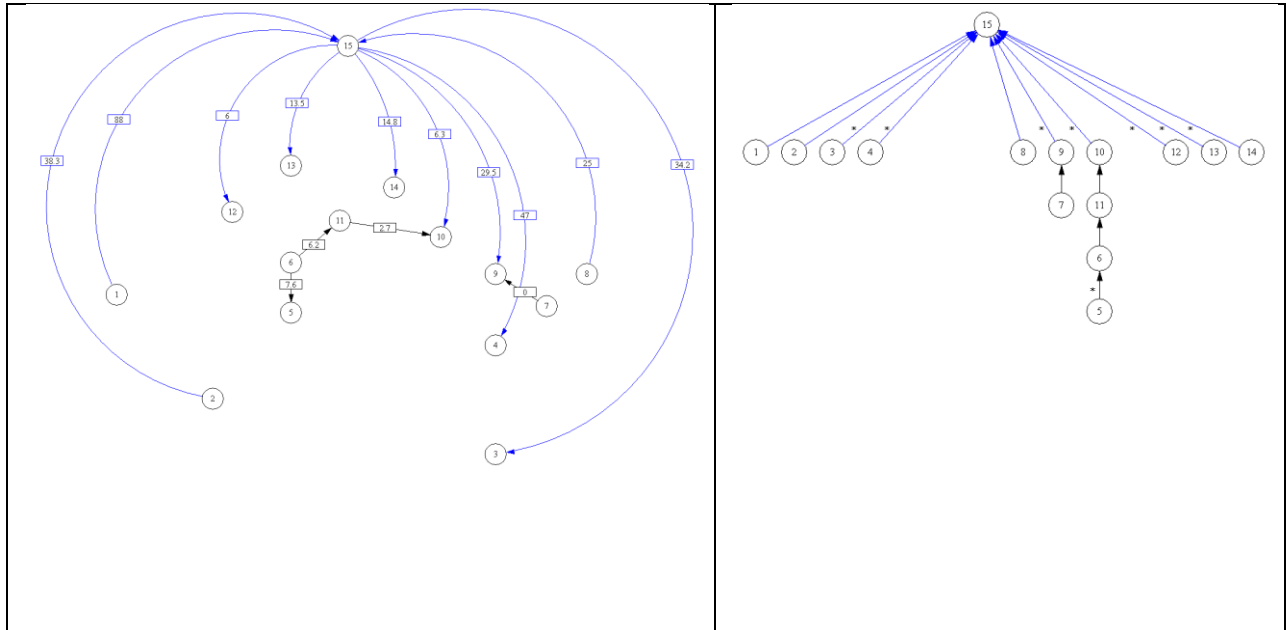


Figure A.9: IEEE 14-Bus System Step 4 Flows.

**A.1.10. IEEE 14-Bus System: Step 5 Cycle**

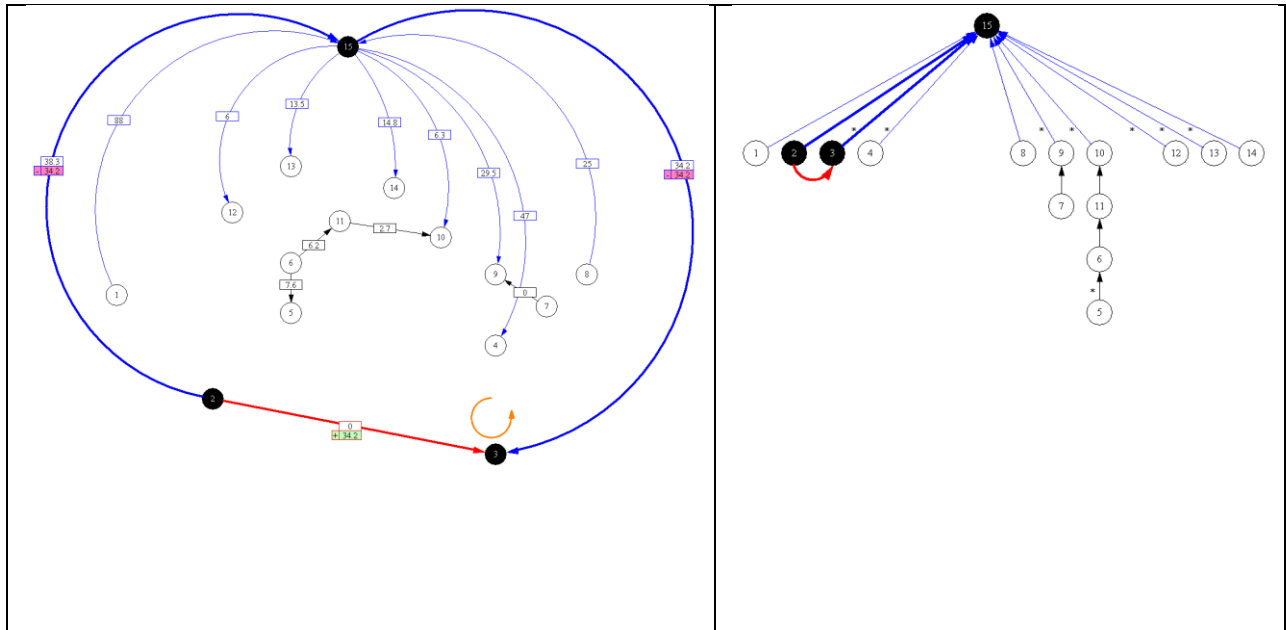


Figure A.10: IEEE 14-Bus System Step 5 Cycle.

**A.1.11. IEEE 14-Bus System: Step 5 Flows**

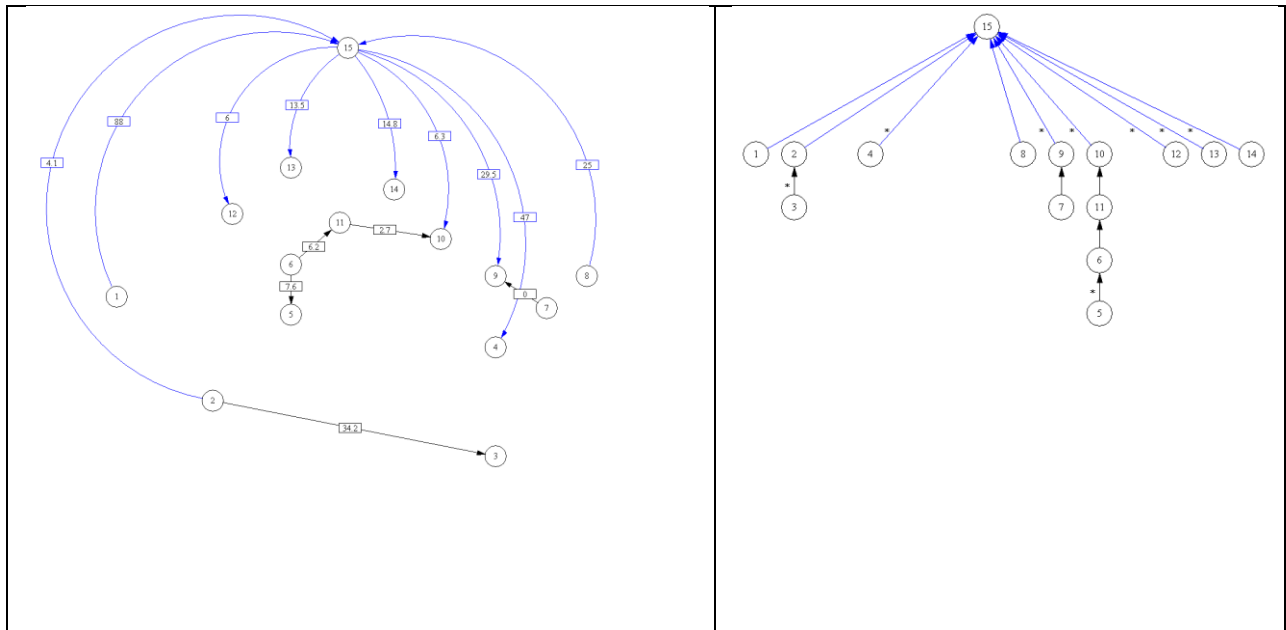


Figure A.11: IEEE 14-Bus System Step 5 Flows.

**A.1.12. IEEE 14-Bus System: Step 6 Cycle**

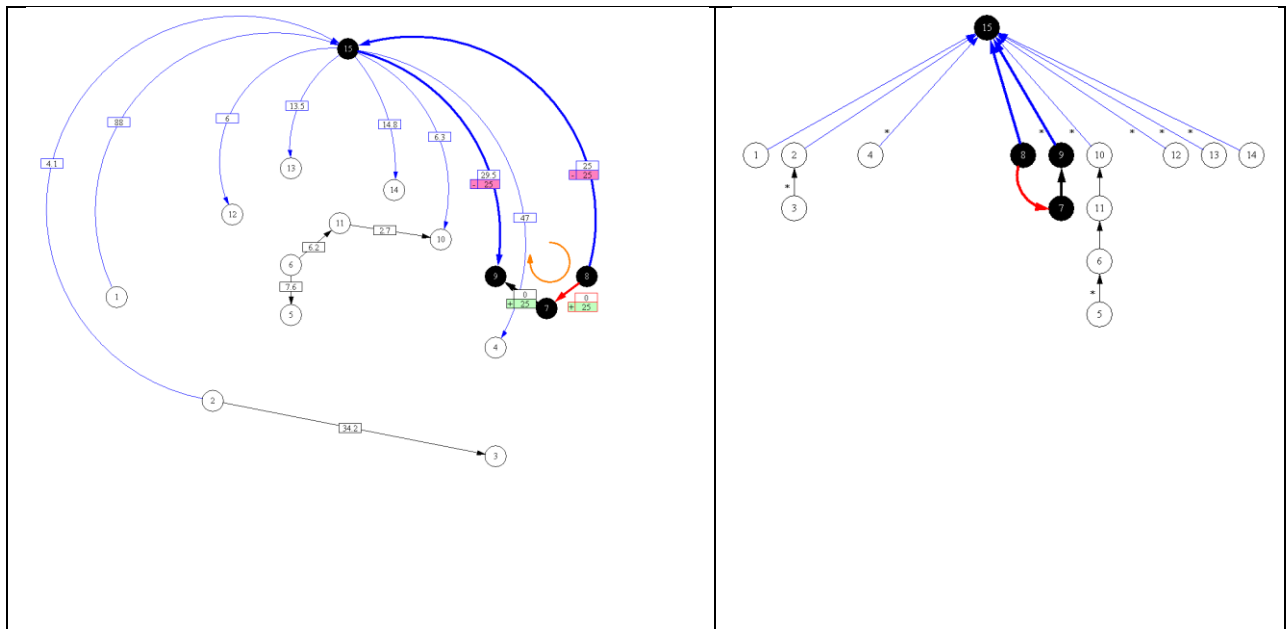


Figure A.12: IEEE 14-Bus System Step 6 Cycle.

**A.1.13. IEEE 14-Bus System: Step 6 Flows**

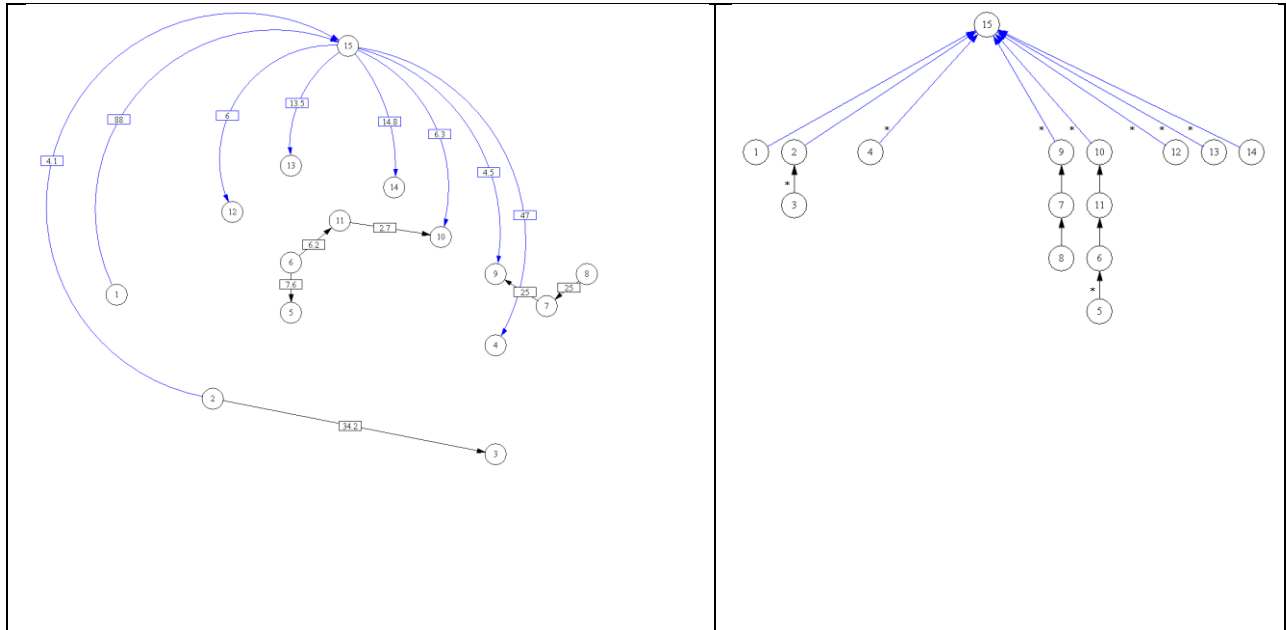


Figure A.13: IEEE 14-Bus System Step 6 Flows.

**A.1.14. IEEE 14-Bus System: Step 7 Cycle**

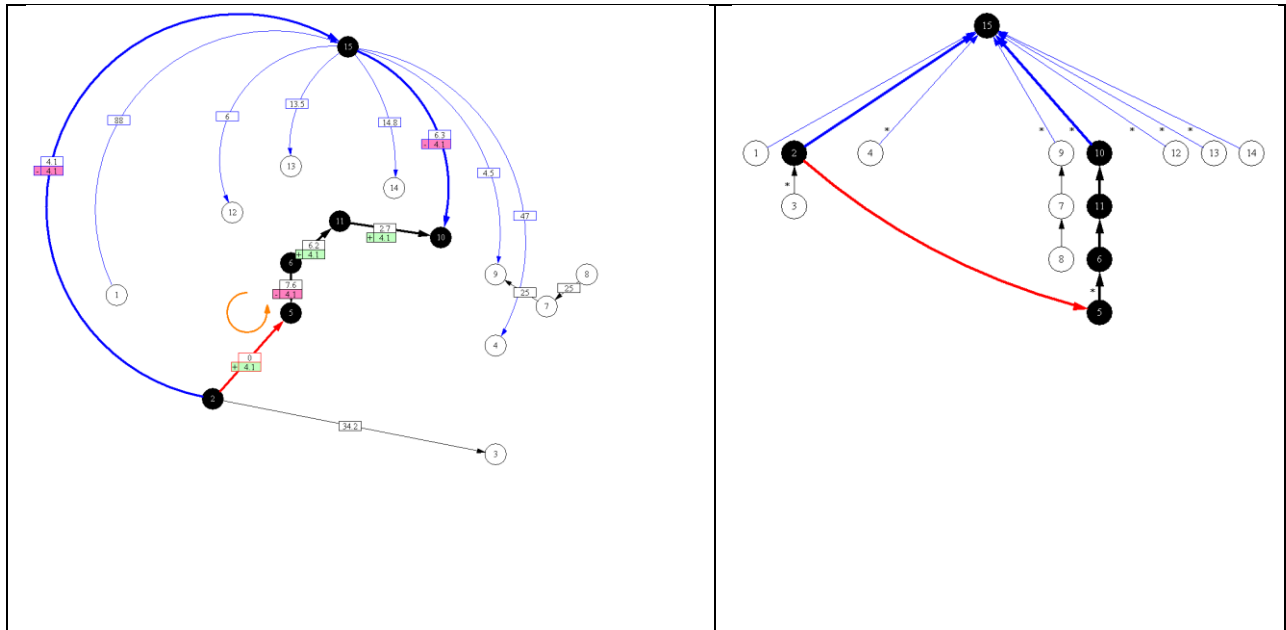


Figure A.14: IEEE 14-Bus System Step 7 Cycle.

**A.1.15. IEEE 14-Bus System: Step 7 Flows**

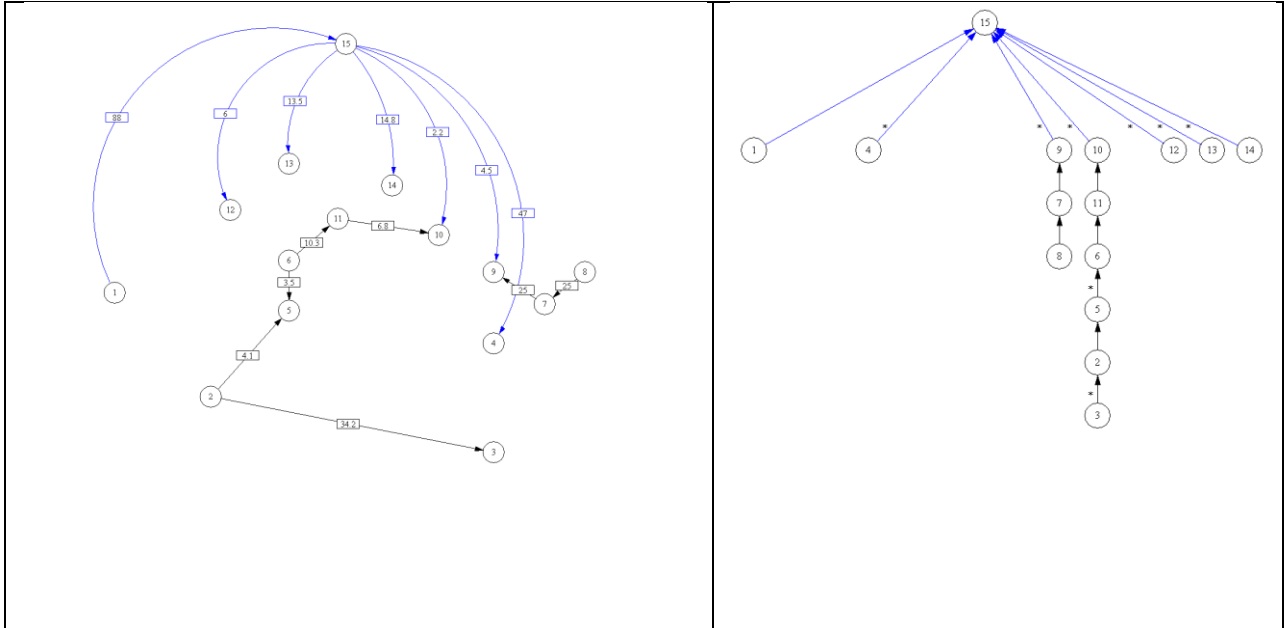


Figure A.15: IEEE 14-Bus System Step 7 Flows.

**A.1.16. IEEE 14-Bus System: Step 8 Cycle**

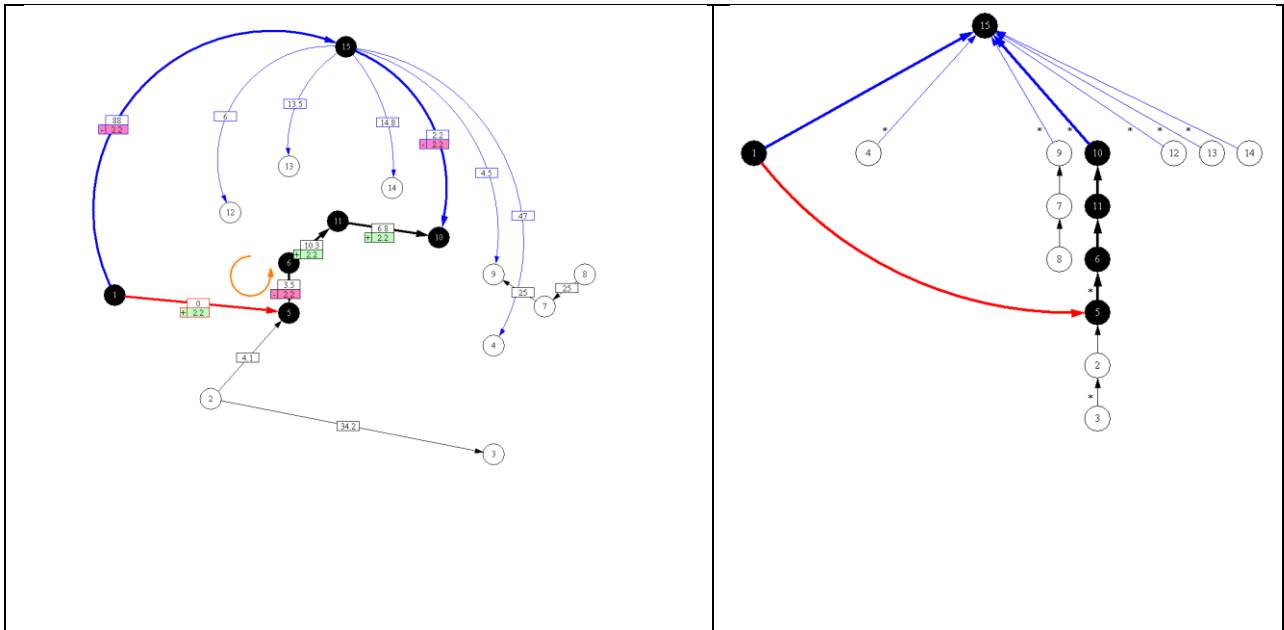


Figure A.16: IEEE 14-Bus System Step 8 Cycle.

**A.1.17. IEEE 14-Bus System: Step 8 Flows**

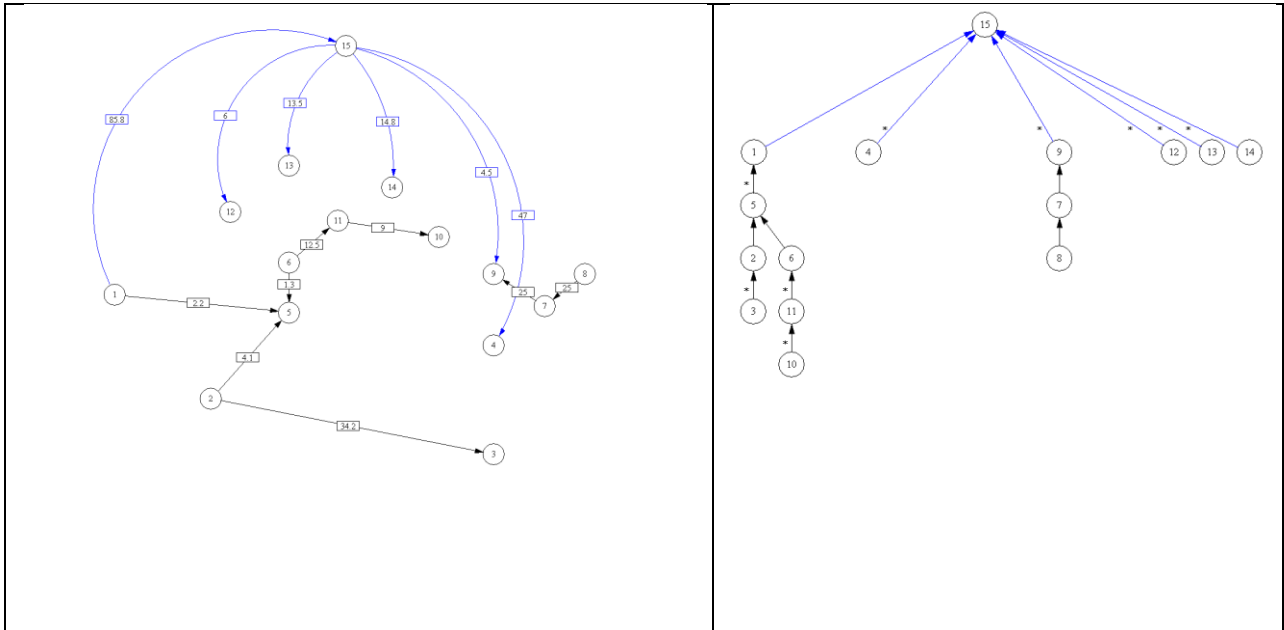


Figure A.17: IEEE 14-Bus System Step 8 Flows.

**A.1.18. IEEE 14-Bus System: Step 9 Cycle**

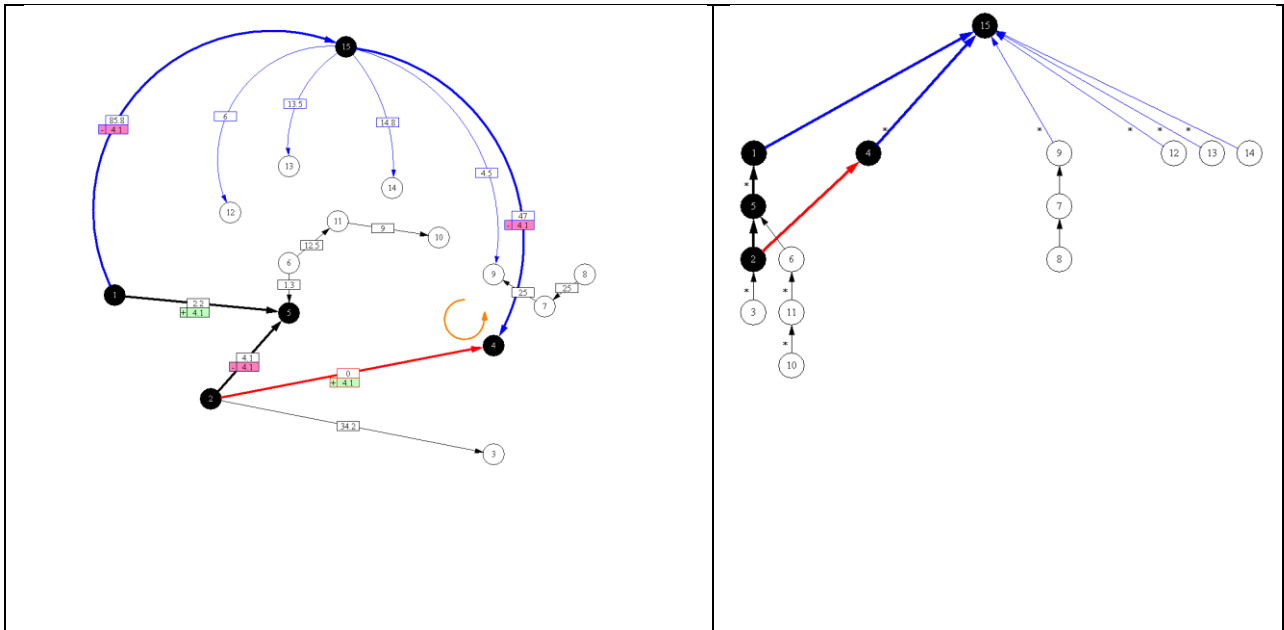


Figure A.18: IEEE 14-Bus System Step 9 Cycle.



**A.1.19. IEEE 14-Bus System: Step 9 Flows**

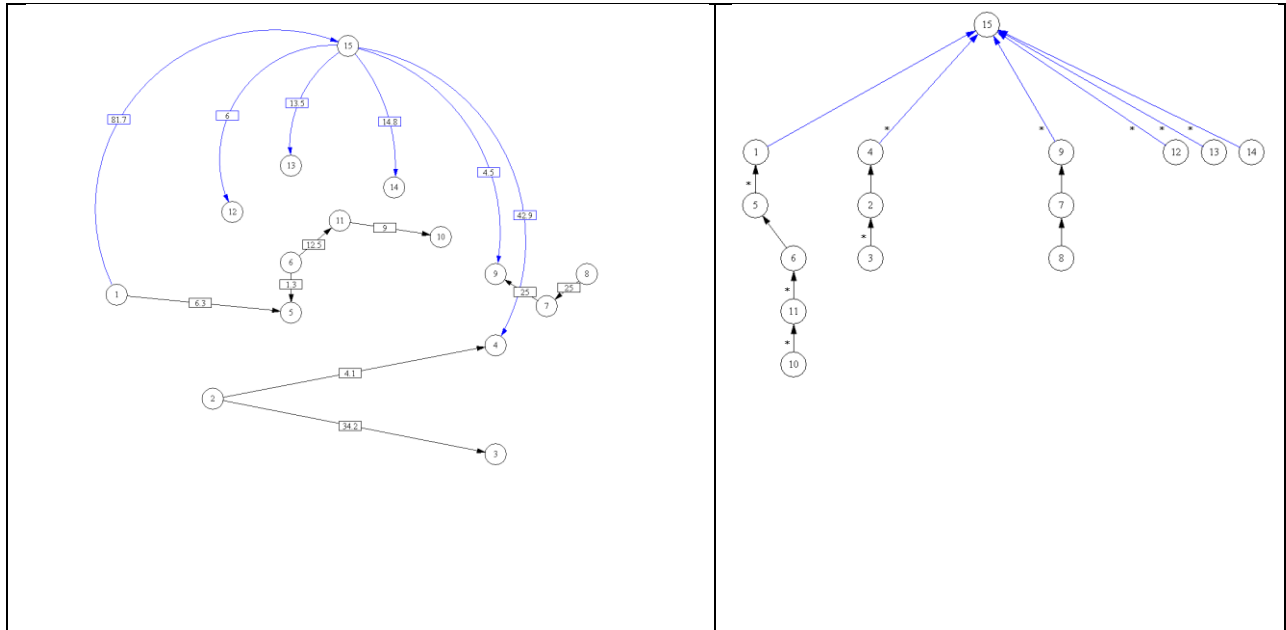


Figure A.19: IEEE 14-Bus System Step 9 Flows.

**A.1.20. IEEE 14-Bus System: Step 10 Cycle**

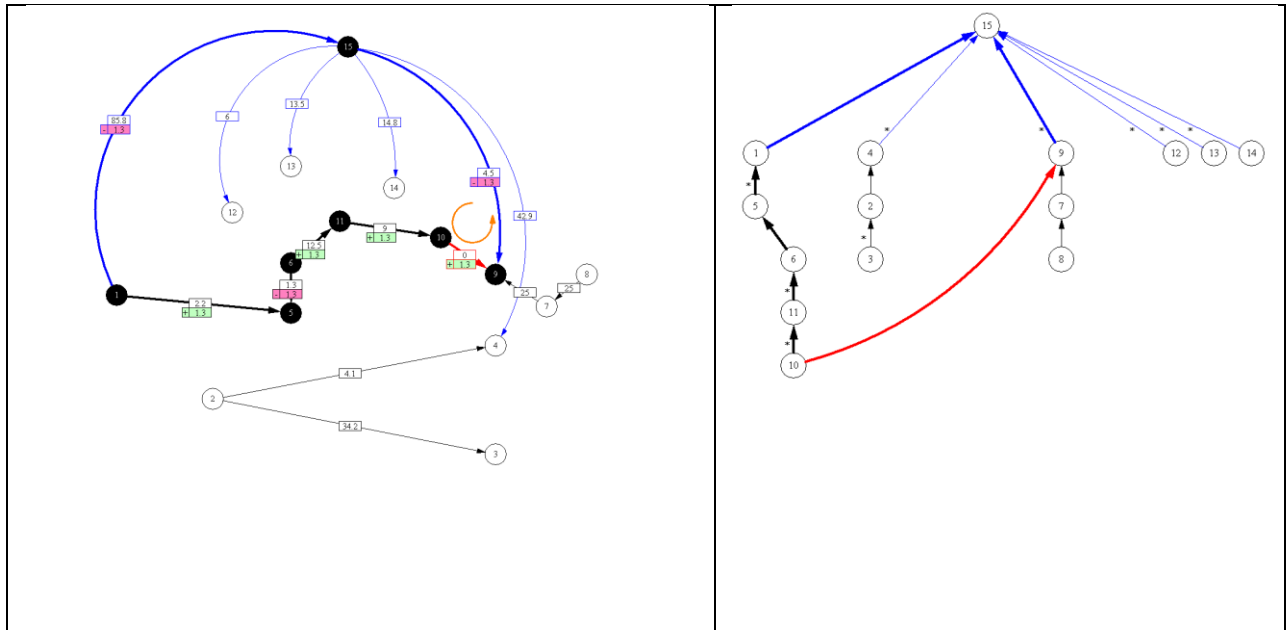


Figure A.20: IEEE 14-Bus System Step 10 Cycle.



**A.1.23. IEEE 14-Bus System: Step 11 Flows**

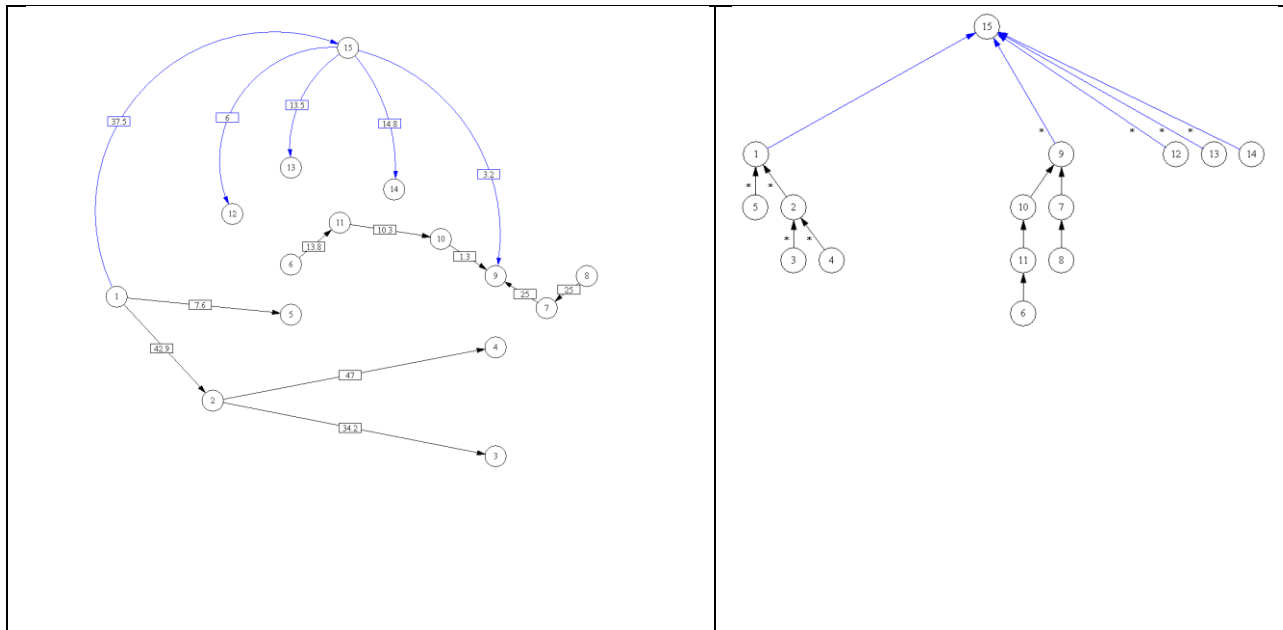


Figure A.23: IEEE 14-Bus System Step 11 Flows.

**A.1.24. IEEE 14-Bus System: Step 12 Cycle**

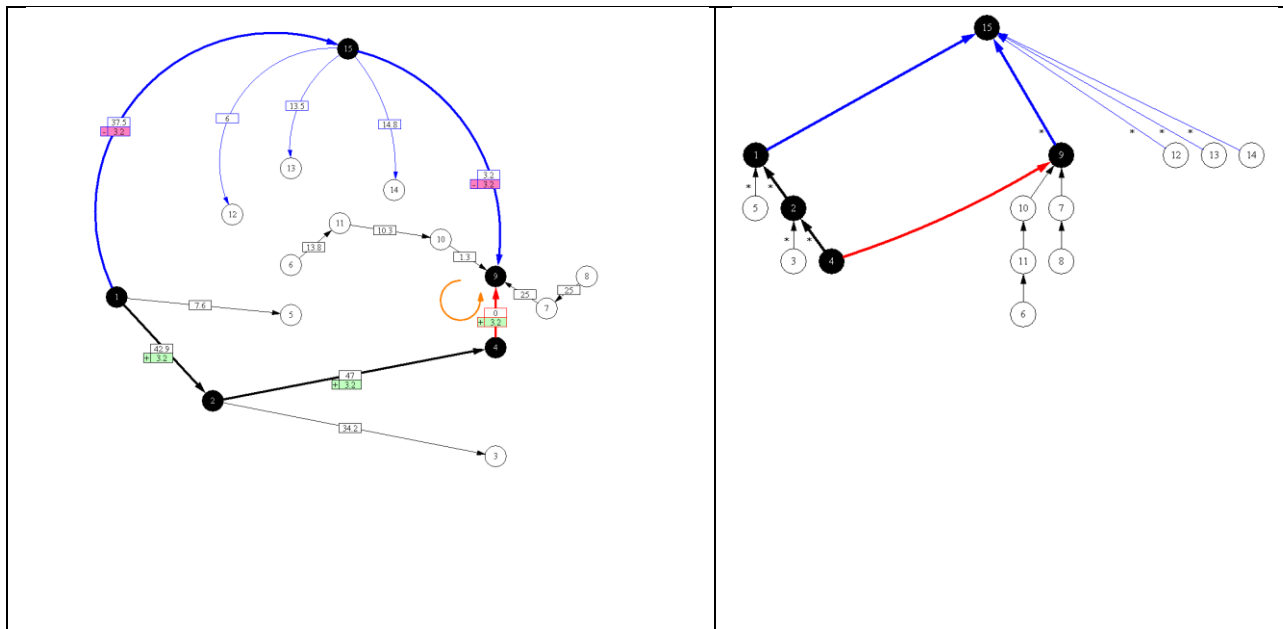


Figure A.24: IEEE 14-Bus System Step 12 Cycle.

**A.1.25. IEEE 14-Bus System: Step 12 Flows**

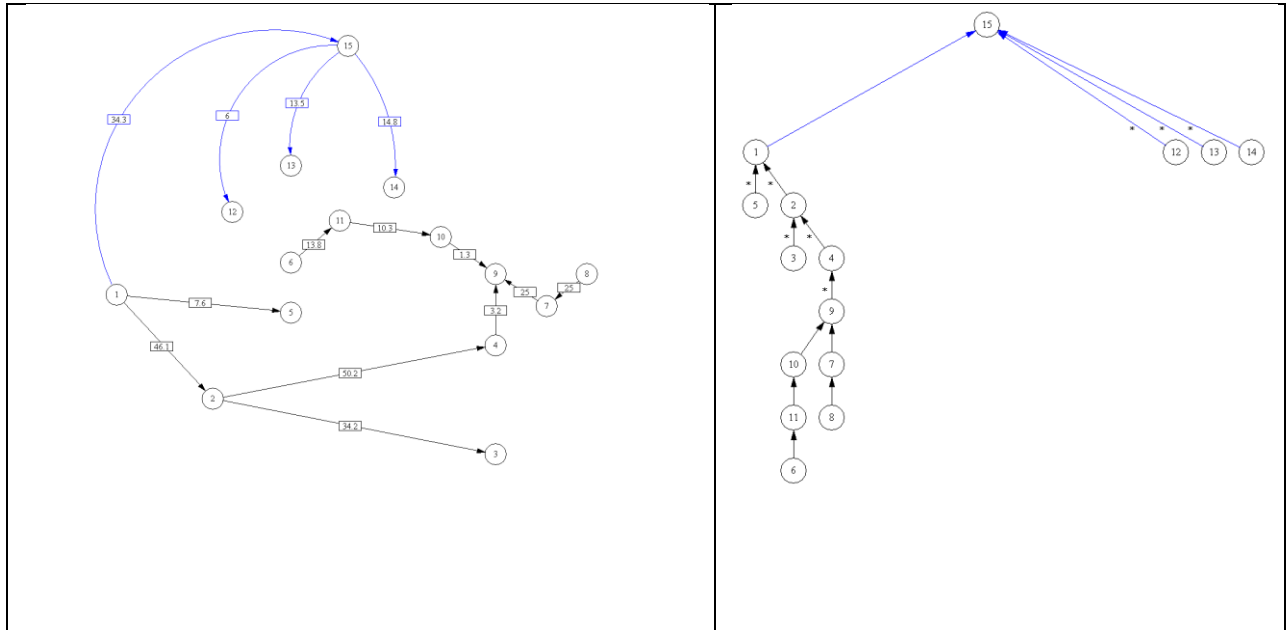


Figure A.25: IEEE 14-Bus System Step 12 Flows.

**A.1.26. IEEE 14-Bus System: Step 13 Cycle**

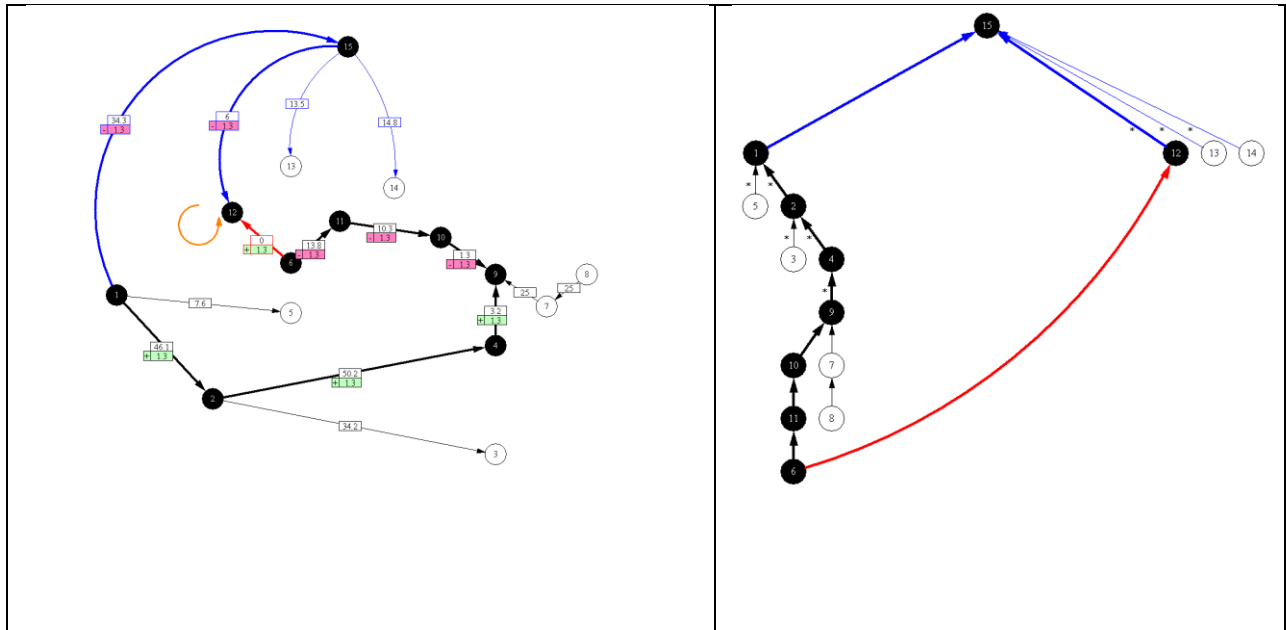


Figure A.26: IEEE 14-Bus System Step 13 Cycle.

**A.1.27. IEEE 14-Bus System: Step 13 Flows**

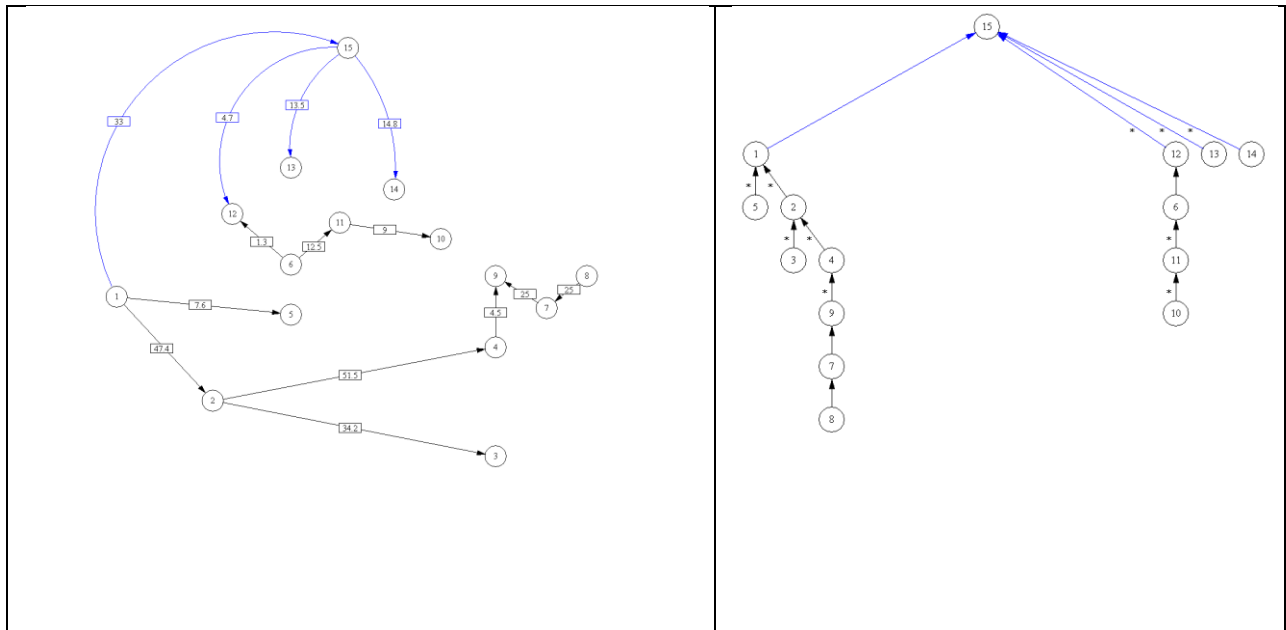


Figure A.27: IEEE 14-Bus System Step 13 Flows.

**A.1.28. IEEE 14-Bus System: Step 14 Cycle**

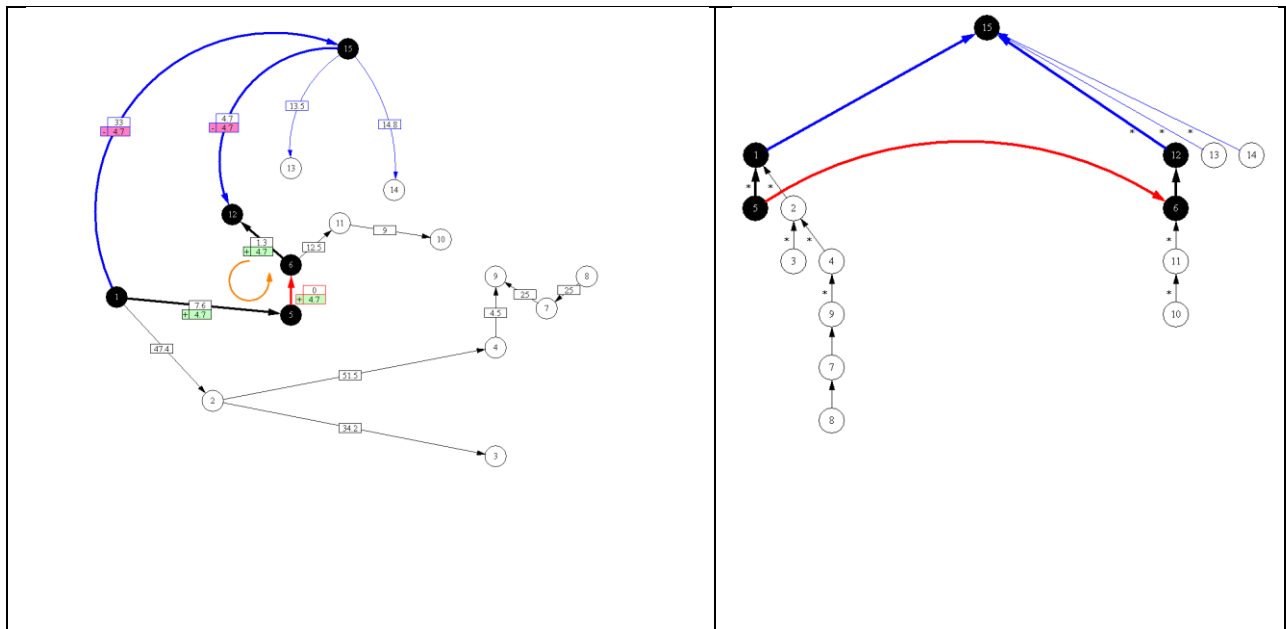


Figure A.28: IEEE 14-Bus System Step 14 Cycle.

**A.1.29. IEEE 14-Bus System: Step 14 Flows**

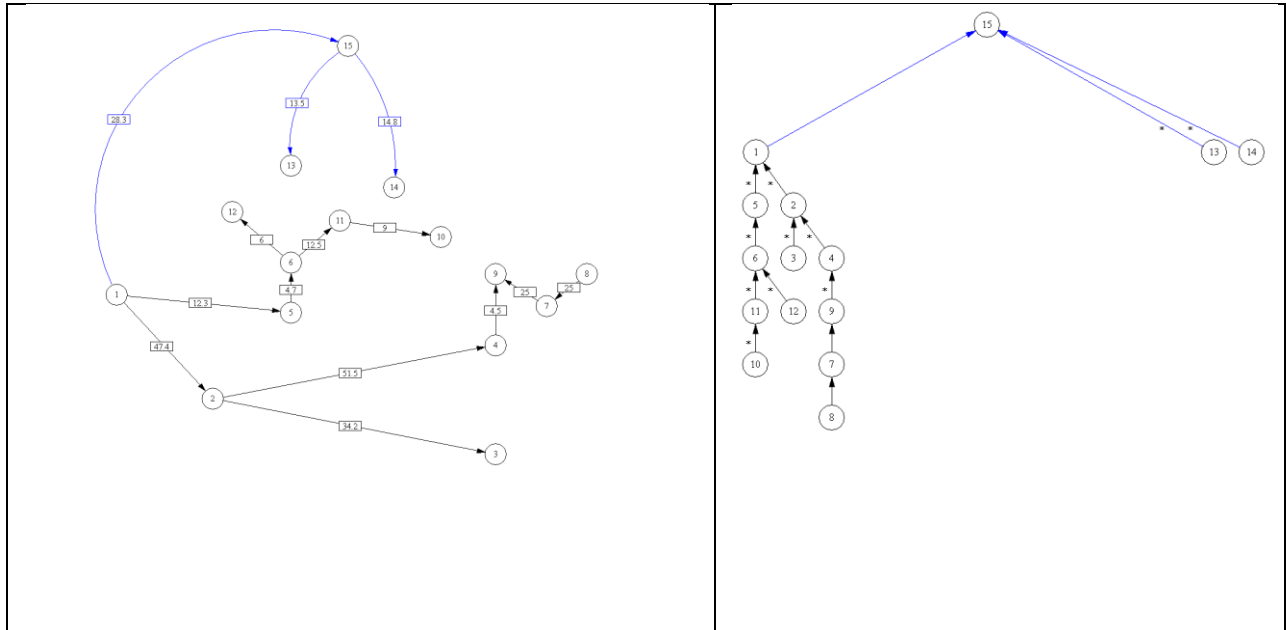


Figure A.29: IEEE 14-Bus System Step 14 Flows.

**A.1.30. IEEE 14-Bus System: Step 15 Cycle**

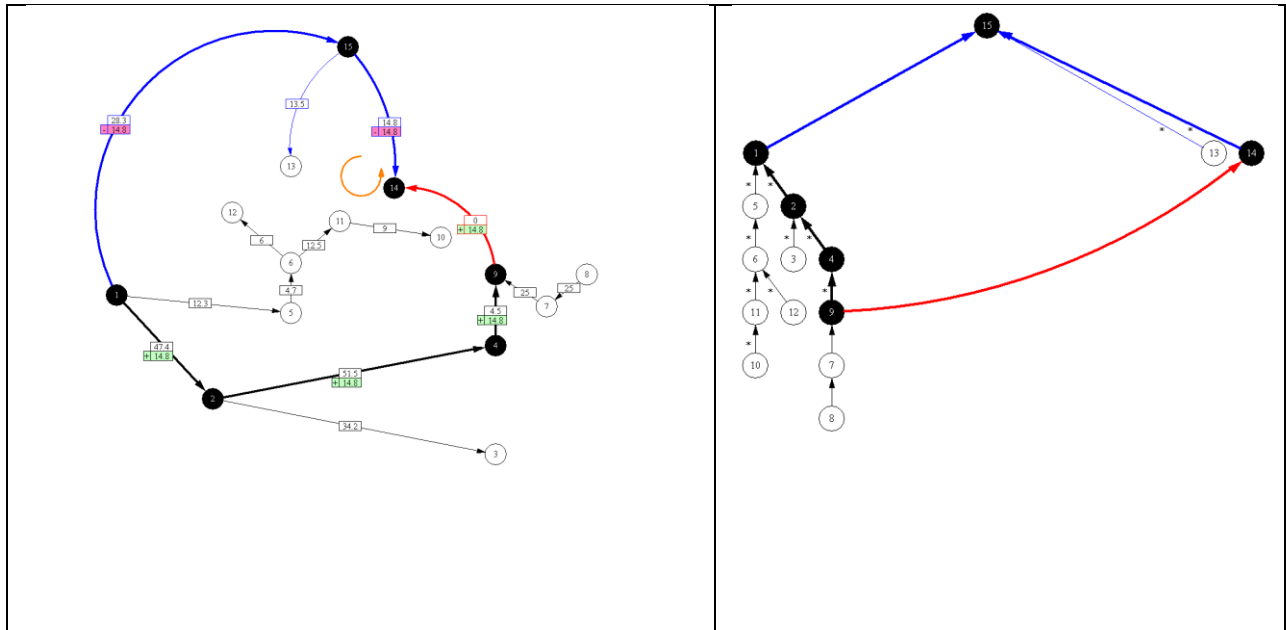


Figure A.30: IEEE 14-Bus System Step 15 Cycle.

**A.1.31. IEEE 14-Bus System: Step 15 Flows**

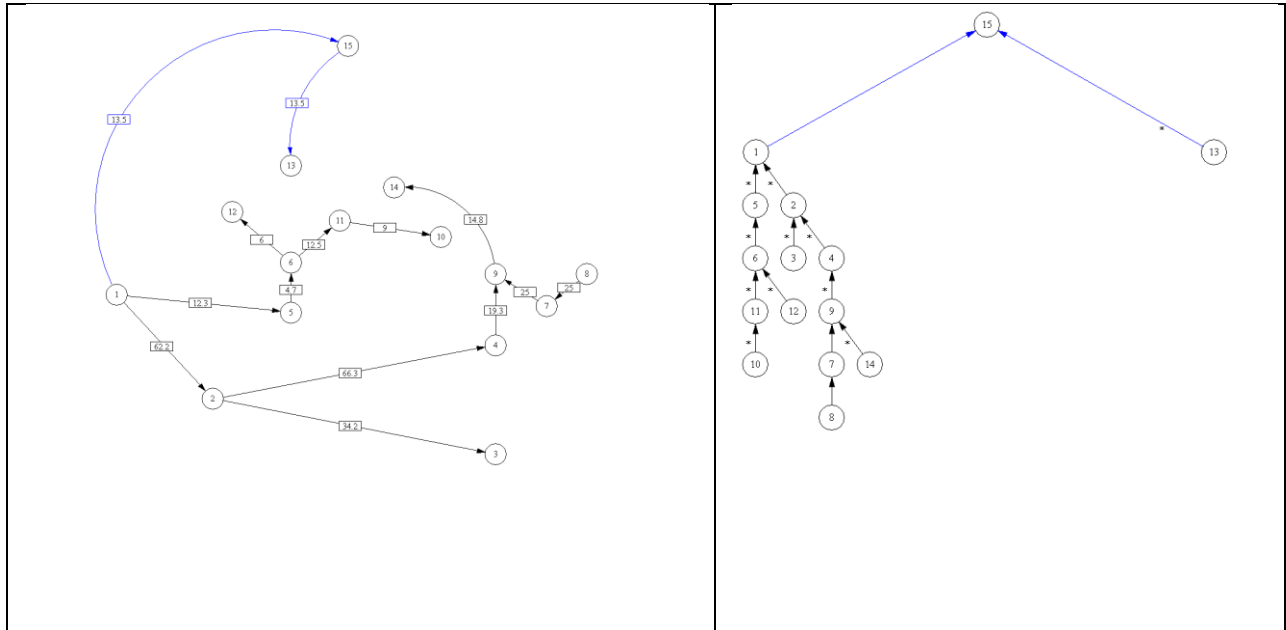


Figure A.31: IEEE 14-Bus System Step 15 Flows.

**A.1.32. IEEE 14-Bus System: Step 16 Cycle**

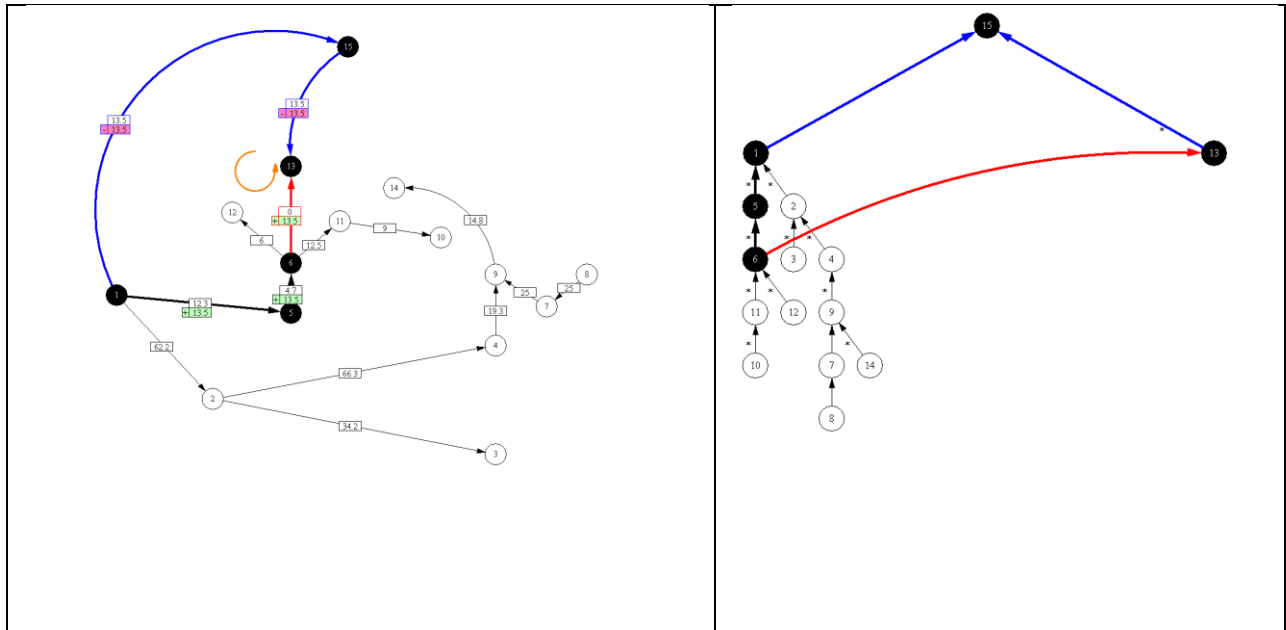


Figure A.32: IEEE 14-Bus System Step 16 Cycle.

**A.1.33. IEEE 14-Bus System: Step 16 Flows (Optimal)**

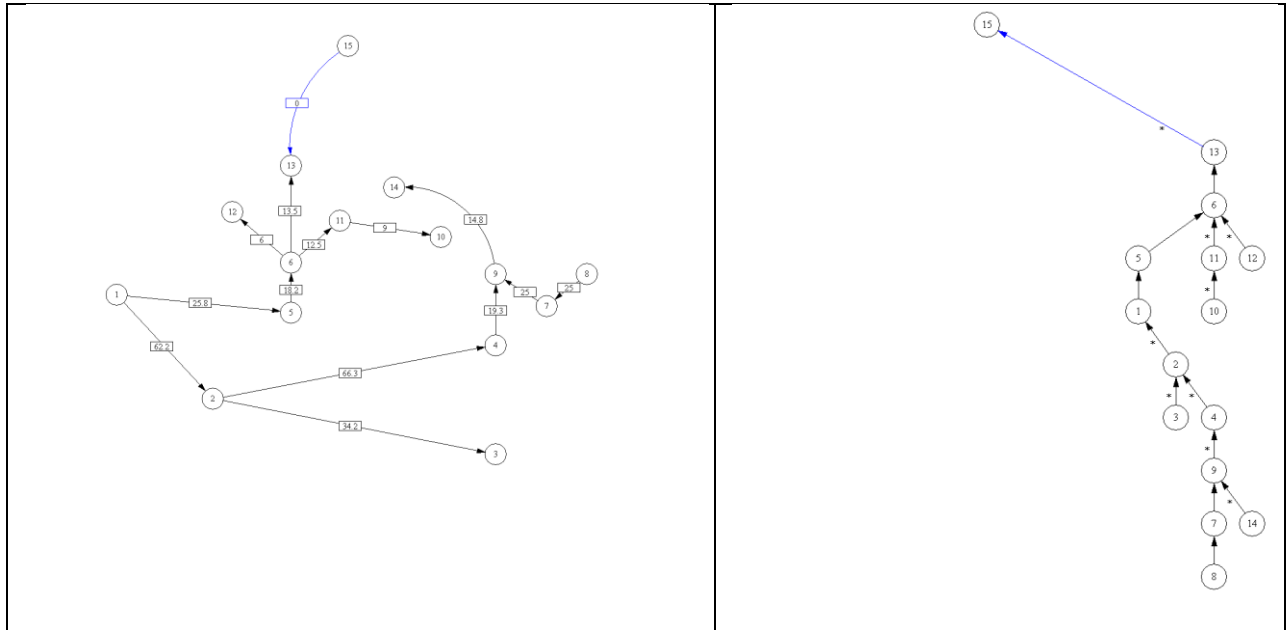


Figure A.33: IEEE 14-Bus System Step 16 Flows (Optimal).