A NETWORK-BASED PRIMAL-DUAL SOLUTION METHODOLOGY FOR THE MULTI-COMMODITY NETWORK FLOW PROBLEM

by

CYNTHIA BARNHART

Bachelor of Science in Civil Engineering
University of Vermont
(1981)

Master of Science in Transportation
Massachusetts Institute of Technology
(1985)

SUBMITTED TO THE DEPARTMENT OF
CIVIL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1988

Signature of Author [illegible]
Department of Civil Engineering
August 5, 1988

Certified by [illegible]
Professor Yosef Sheffi
Head, Transportation Systems Division
Thesis Supervisor

Certified by [illegible]
Professor Ole S. Madsen
Chairman, Departmental Committee on Graduate Students
Department of Civil Engineering

A NETWORK-BASED PRIMAL-DUAL SOLUTION METHODOLOGY FOR THE MULTI-COMMODITY NETWORK FLOW PROBLEM

by

CYNTHIA BARNHART

Submitted to the Department of Civil Eng' ıeering
on August 5, 1988
in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Civil Engineering

ABSTRACT

The Capacitated Multi-Commodity Network Flow (CMCF) problem arises whenever commodities (goods, people, vehicles, etc.) must be transported from one location to another over a network with finite capacity. The objective is to determine the cost-minimizing assignment of commodities to arcs such that each commodity is transported from its origin to its destination without violating capacity restrictions. Numerous applications of this problem type can be found in the transportation industry, particularly in the areas of vehicle routing and scheduling with freight flow assignment. These real-world problems can be extremely large, however, and thus, are unsolvable using existing solution procedures. As a result, a new algorithm, called PDN, was developed specifically for the solution of large-scale CMCF problems.

The PDN algorithm is cast in the primal-dual framework and uses a network-based solution strategy to solve the CMCF problem. The network-based procedure, by eliminating the restrictions on problem size imposed by the simplex method, provides the PDN algorithm with the flexibility to solve large-scale problems.

Computational tests show that the PDN algorithm is able to solve problems of much larger dimension than could previously be solved using existing solution techniques. Furthermore, the PDN algorithm is more efficient than the simplex-based primal-dual algorithm in solving even small to moderate size problems.

Thesis Supervisor: Dr. Yosef Sheffi
Title: Professor of Civil Engineering
Head, Transportation Systems Division

ACKNOWLEDGEMENTS

I would like to thank the sponsors of my research work, namely North American Van Lines (NAVL) and the Center for Transportation Studies at M.I.T. Thanks to NAVL for introducing me to the complexities of "real-world" problems and providing me with an interesting research topic.

Thanks to Professor Yosef Sheffi for his guidance and support during my stay at MIT. Thanks also to Professors Tom Magnanti, Jim Orlin, Warren Powell, Hari Psaraftis, and Jerry Shapiro for their help in my research work.

Special, special thanks to all my friends who made my stay at M.I.T. an experience I will not forget. Thanks to Tom Walsh (how far can long-sleeves be rolled up anyway?), Ursula McCord, Maria Marangiello, Thierry "I ate Central Square" Mallet, Babak Eskandari (or is it Eskandari Babak?), and of course, my pals-- Ilana Berger, Rina Rotshild, and Teresa Valdivieso. Last but certainly not least, thanks to Dr. Haris Koutsopoulos for all his helpful advise and encouragement.

Finally, thanks to my family for supporting me in this endeavor. Thanks especially to Mark for always being there.

TABLE OF CONTENTS

Page

Abstract.. 2

Acknowledgements.. 3

Table of Contents... 4

List of Figures... 10

List of Tables.. 14

1. Introduction... 15

1.1 Problem Description................................... 15

1.2 Literature Review..................................... 18

1.2.1 Partitioning Methods................................ 19

1.2.2 Resource-Directive Decomposition.................... 19

1.2.3 Price-Directive Decomposition....................... 20

1.2.4 Other Solution Techniques........................... 23

1.3 Computational Experience.............................. 24

1.4 Thesis Outline.. 26

2. The Primal-Dual Method and the CMCF Problem............ 28

2.1 Path Formulation of the CMCF Problem.................. 28

2.2 Primal-Dual Method.................................... 31

2.2.1 Step 0: Generate Initial Solution................... 35

2.2.2 Step 1: Define Admissible Elements.................. 35

2.2.3 Step 2: Solve the Restricted Primal and Dual Problem.. 36

Page

2.2.4 Step 3: Test for Optimality.................... 40

2.2.5 Step 4: Generate a Feasible Solution for D....... 42

2.2.5.1 Dual Price Adjustments, Admissibility and Dual Feasibility........................ 43

2.2.5.2 Dual Price Adjustments and the Dual Objective Function Value.................... 49

2.3 Primal-Dual Algorithm-- A Summary.................... 50

2.4 Finiteness.. 51

2.5 Summary... 54

3. Algorithmic Design.................................. 55

3.1 Algorithmic Motivation............................ 55

3.2 Primal-Dual Method vs. Dantzig-Wolfe Method........... 56

3.3 Algorithmic Framework............................. 59

3.4 Algorithmic Design Rules and Criterion................ 62

3.5 Summary... 64

4. Solution Method for the RP and DRP Problems............... 65

4.1 The RP and DRP Formulations Restated.................. 65

4.2 Definitions.. 67

4.3 Search Process..................................... 72

4.4 Arc Costs Revisited................................ 82

4.4.1 Negative Cost Cycles in RN...................... 87

4.4.2 Positive Cost Cycles in RN...................... 89

Page

4.5 Multi-Commodity Effect on Problem Complexity........... 91

4.6 Summary... 93

5. The Flow Adjustment Algorithm.......................... 95

5.1 Overview and Example................................ 95

5.2 Steps of the Flow Adjustment Algorithm.............. 99

5.2.1 FAA Step 0: Initialize........................... 101

5.2.2 FAA Step 1: Select Cyclearc ij and Cycle Γ from Stack S.................................. 101

5.2.3 FAA Step 2: Search for Negative Cost Cycle....... 102

5.2.4 Faa Step 3: Update............................... 103

5.2.5 FAA Step 4: Compute ΔI Measure.................. 103

5.2.6 FAA Step 5: Select New Cyclearc ij from Cycle Γ and Update............................... 104

5.2.7 FAA Step 6: Update Cyclearc Cost................ 106

5.2.8 FAA Step 7: Shift Flows.......................... 108

5.2.9 FAA Step 8: Remove Cyclearc ij and Cycle Γ Pair from Stack S.............................. 112

5.2.10 FAA Step 9: Compute Ψ_Γ......................... 112

5.2.11 FAA Step 10: Compute Change in ΔI due to Cycle Γ.. 112

5.2.12 FAA Step 11: Remove Cycle Shifts from List Ω..... 113

5.2.13 FAA Step 12: Update Stack S..................... 114

5.2.14 FAA Step 13: Test if Terminate.................. 114

Page

5.3 Termination of the Flow Adjustment Algorithm........... 115

5.4 Finiteness and Flow Adjustment Steps.................. 120

5.5 Cycling and Flow Adjustment Steps..................... 126

5.6 Summary... 130

6. The PDN Algorithm...................................... 132

6.1 PDN Step 0: Initialization............................ 133

6.2 PDN Step 1: Define Admissible Elements................ 134

6.3 PDN Step 2: Solve RP and DRP Problem.................. 134

6.4 PDN Step 3: Test for Optimality....................... 135

6.5 PDN Step 4: Generate New Solution for D............... 135

6.5.1 Satisfaction of Design Rules and Criterion........ 136

6.5.2 Implementation.... 137

6.6 Algorithm... 140

6.7 Finiteness.. 141

6.8 Summary... 144

7. Experimental Design.................................... 146

7.1 LP Solution Methodology............................... 146

7.2 DW Solution Methodology............................... 147

7.3 PD Solution Methodology............................... 148

7.4 PDN Solution Methodology.............................. 150

7.5 Test Data... 153

7.6 Computing Environment................................. 156

Page

7.7 Performance Measures.............................. 157

7.7.1 Number of Iterations.............................. 157

7.7.2 Run Time.............................. 158

8. Computational Results.............................. 160

8.1 Run Time vs. Number of Iterations.............................. 161

8.2 Run Time vs. Total Demand.............................. 162

8.3 Run Time vs. Number of Demands.............................. 164

8.4 Run Time vs. Average Demand Size.............................. 166

8.5 Run Time vs. Flow Shift Criterion.............................. 168

8.6 LP Solution Methodology.............................. 170

8.7 PDN Solution Methodology.............................. 173

8.7.1 PDN1 vs. PDN2 Implementations.............................. 174

8.7.2 PDN1 vs. PDN2: Number of Iterations.............................. 175

8.7.3 PDN1 vs. PDN2: Initialization Times.............................. 177

8.7.4 PDN1 vs. PDN2: Dual Times.............................. 177

8.7.5 PDN1 vs. PDN2: RP Solution Times.............................. 177

8.7.6 PDN1 vs. PDN2: Total Run Time.............................. 180

8.8 PD Solution Methodology.............................. 181

8.8.1 PD1 vs. PD2 Implementations.............................. 181

8.8.2 PD1 vs. PD2: Number of Iterations.............................. 181

8.8.3 PD1 vs. PD2: Initialization Times.............................. 187

8.8.4 PD1 vs. PD2: Dual Times.............................. 187

8.8.5 PD1 vs. PD2: RP Solution Times.............................. 189

Page
8.8.6 PD1 vs. PD2: Total Run Time...................... 190
8.9 PDN vs. PD Implementations............................ 191
8.9.1 PDN vs. PD: Number of Iterations................ 192
8.9.2 PDN vs. PD: Initialization Times................ 195
8.9.3 PDN vs. PD: Dual Times.......................... 196
8.9.4 PDN vs. PD: RP Solution Times................... 196
8.9.5 PDN vs. PD: Total Run Time...................... 196
8.10 DW vs. PDN and PD Implementations.................... 198
8.10.1 DW vs. PDN and PD: Number of Iterations......... 198
8.10.1.1 DW vs. PDN and PD: Path Generation......... 201
8.10.1.2 DW vs. PDN and PD: Tailing-Off Effect...... 206
8.10.2 DW vs. PDN and PD: Initialization Times......... 206
8.10.3 DW vs. PDN and PD: Dual Times.................. 207
8.10.4 DW vs. PDN and PD: RP Solution Times............ 208
8.10.5 DW vs. PDN and PD: Total Run Time............... 208
8.11 Solution of Large-Scale Problems..................... 211
8.12 Conclusions.. 216
9. Conclusions... 219
9.1 Contributions... 219
9.2 Future Research....................................... 221
9.2.1 Network-Based Dantzig-Wolfe Algorithm............ 222
9.2.2 Preprocessing Dual Ascent Step for PDN Algorithm.. 226
9.2.3 Relaxation Strategy and Parallel Computation...... 228
Bibliography.. 229

LIST OF FIGURES

Page

Chapter 2

Figure 2.1: Primal-Dual Method 35

Figure 2.2: Primal-Dual Algorithm 52

Chapter 3

Figure 3.1: The PDN Algorithm 61

Figure 3.2: Solution of the RP/DRP Problem 62

Chapter 4

Figure 4.1: Cycle Flow Shift 73

Figure 4.2: Composite Flow Shift 80

Figure 4.3a: Cycling in Negative Cost Cycle Search--
Part I 88

Figure 4.3b: Cycling in Negative Cost Cycle Search--
Part II 88

Figure 4.4: Equivalent Single Cycle 92

Chapter 5

Figure 5.1: Domino-Effect 98

Figure 5.2: The Flow Adjustment Algorithm 100

Figure 5.3: Determination of the Optimal Flow Shift Size.. 111

Figure 5.4a: Infinite Sequence of Flow Adjustment Steps--
Part I 121

Page

Chapter 5 (cont.)

Figure 5.4b: Infinite Sequence of Flow Adjustment Steps--
Part II 122

Figure 5.4c: Infinite Sequence of Flow Adjustment Steps--
Part III 123

Figure 5.4d: Infinite Sequence of Flow Adjustment Steps--
Part IV 124

Figure 5.5b: Cycling in the Flow Adjustment Algorithm--
Part I 126

Figure 5.5b: Cycling in the Flow Adjustment Algorithm--
Part II 127

Figure 5.5c: Cycling in the Flow Adjustment Algorithm--
Part III 128

Figure 5.5d: Cycling in the Flow Adjustment Algorithm--
Part IV 129

Chapter 6

Figure 6.1: The PDN Methodology 132

Figure 6.2: Pseudo-code for the PDN Algorithm 142

Chapter 8

Figure 8.1: Run Time and Number of Iterations (P1 & P3)... 161

Figure 8.2: Run Time and Number of Iterations (P2 & P4)... 162

Page

Chapter 8 (cont.)

Figure 8.3: Run Time, Total Demand and Number of Demands (P1 & P3) 163

Figure 8.4: Run Time, Total Demand and Number of Demands (P2 & P4) 164

Figure 8.5: Number of Iterations and Total Demand 165

Figure 8.6: Dual Time Per Iteration and Number of Commodities 166

Figure 8.7: RP Solution Time and Average Demand 168

Figure 8.8: Average Total Number of Flow Shifts 170

Figure 8.9: PDN RP Solution Time 171

Figure 8.10: Run Time Ratio: LP vs. All Others 172

Figure 8.11: PDN Run Time by Components 174

Figure 8.12: PDN Mean Number of Iterations 175

Figure 8.13: PDN Dual Objective Function Ascent 176

Figure 8.14: PDN Dual Time per Iteration 178

Figure 8.15: PD Run Time by Components 182

Figure 8.16: Mean Number of PD Iterations 183

Figure 8.17: PD Dual Objective Function Ascent 184

Figure 8.18: Example Network 184

Figure 8.19: PD1 Solution Approach 185

Figure 8.20: PD2 Solution Approach 186

Figure 8.21: Number of Columns Added Per Iteration 189

Page

Chapter 8 (cont.)

Figure 8.22: PDN and PD Time by Components................. 191

Figure 8.23: PDN vs. PD: Dual Objective Function Ascent... 192

Figure 8.24: PDN vs. PD: Run Time, Number of Iterations and Total Demand................... 194

Figure 8.25: PDN vs. PD vs. DW: Run Time by Components.... 199

Figure 8.26: PDN, PD & DW: Number of Iterations and Total Demand.................................. 200

Figure 8.27: Initial Set of Shortest Paths................. 203

Figure 8.28: PD and PDN Solution Methodologies............. 203

Figure 8.29: DW Solution Methodology...................... 204

Figure 8.30: Tailing-Off Effect........................... 207

Figure 8.31: Problem P1: Run Time by Components........... 209

Figure 8.32: Problem P4: Run Time by Components........... 210

Figure 8.33: Freight Assignment Example................... 214

LIST OF TABLES

Page

Chapter 1

Table 1.1: Size of Problems Solved........................ 25

Chapter 3

Table 3.1: Primal-Dual vs. Dantzig-Wolfe.................. 58

Chapter 7

Table 7.1: Variations on the PDN Algorithmic Implementations................................ 153

Table 7.2: Characteristics of the Test Problems........... 154

Table 7.3: Variations within Test Problems................ 155

Chapter 8

Table 8.1: PDN1 vs. PDN2.................................. 180

Table 8.2: PD1 vs. PD2.................................... 190

Table 8.3: PDN vs. PD..................................... 197

Table 8.4: DW vs. PD/PDN.................................. 211

1. Introduction

This chapter introduces the **Capacitated Multi-Commodity Network Flow Problem** (CMCF) and then reviews the relevant literature. Section 1.1 first presents the CMCF problem description and its mathematical formulation. Then, Sections 1.2 and 1.3 describe existing solution techniques for the **linear** CMCF problem. Finally, Section 1.4 outlines the general structure of this thesis.

1.1 Problem Description

The Capacitated Multi-Commodity Network Flow (CMCF) Problem arises whenever commodities (goods, people, vehicles, etc.) must be transported from origin to destination locations over a network with finite capacity. The multi-commodity problem can be conceptualized as a set of single-commodity problems tied together by "bundle" constraints. Each single-commodity problem requires the transport of one commodity from its origin to its destination, conserving flow along the way. The bundle constraints ensure that the commodities, which flow along the same arcs, share arc capacity without exceeding the amount available. The CMCF objective is to determine the assignment of commodities to arcs such that the total cost of transporting all commodities is minimized.

These objective and constraints can be represented for the linear CMCF problem in node-arc form as follows:

$$z^*_P = \text{MIN } \Sigma_{ij\in A} \Sigma_{k\in K} c_{ij}^k x_{ij}^k \qquad [1.1]$$

subject to:

$$\Sigma_{k\in K}\, x_{ij}{}^{k} + s_{ij} = d_{ij}, \quad \forall\ ij\in A \qquad [1.2]$$

$$\Sigma_{j\in N}\, x_{ij}{}^{k} - \Sigma_{i\in N}\, x_{ij}{}^{k} = \begin{cases} b^{k} \text{ if } i = s^{k}, \\ -b^{k} \text{ if } i = t^{k}, \\ 0, \text{ otherwise}; \ \forall i\in N,\ \forall k\in K. \end{cases} \qquad [1.3]$$

$$x_{ij}{}^{k} \geq 0,\ \forall\ ij\in A,\ \forall\ k\in K; \quad s_{ij} \geq 0,\ \forall\ ij\in A. \qquad [1.4]$$

where:

K: set of 1,...,$|K|$ commodities;

A: set of arcs;

N: set of nodes;

d_{ij}: = capacity of arc $ij\in A$;

$x_{ij}{}^{k}$: quantity of flow of commodity $k\in K$ on arc $ij\in A$;

$c_{ij}{}^{k}$: per unit cost of flow on arc ij of commodity k;

b^{k}: quantity of commodity $k\in K$ supplied at origin; ($-b^{k}$: quantity of commodity $k\in K$ demanded at destination);

s_{ij}: slack variables for bundle constraints [1.2]-- representing the amount of unused arc capacity, i.e.: $s_{ij} = d_{ij} - X_{ij}$; where $X_{ij} = \Sigma_{k\in K}\, x_{ij}{}^{k}$;

The node-arc formulation of the CMCF problem exhibits block diagonal form with coupling rows. Equations [1.2] are the bundle or arc capacity constraints which limit the total flow on each arc to the capacity of that arc. Equations [1.3] are the conservation of flow constraints which ensure for each commodity, that the total quantity demanded is transported from the origin to the destination. Finally, equations [1.4] are the flow and slack non-negativity constraints. The objective, equation [1.1], is to minimize total cost.

The CMCF problem presents itself in several transportation applications. For example, the following transportation-related problems have been formulated as CMCF problems:

i) tanker scheduling [9];

ii) urban traffic assignment [33];

iii) school desegregation [16];

iv) routing freight cars over a railroad [56,57]; and

v) design of a route structure for an Air Force air cargo system [2].

The freight assignment problem requires the transport of shipments from their origins to their destinations using a fleet of vehicles. This problem can be formulated as a linear CMCF problem where:

i) the objective is to minimize total service time for the freight; and

ii) the underlying network represents the possibilities for freight movement.

Freight (or personnel) assignment problems must be repeatedly solved as part of the everyday operations of truck, ship, rail or air transportation companies. However, primarily because real-world problem sizes tend to be extremely large, many of these freight assignment problems are very difficult to solve. Hence, it is concluded that the transportation industry could benefit from a solution methodology geared specifically to the solution of large-scale CMCF problems.

This observation motivates the research work described in this thesis.

Although the **linear** CMCF problem is the focus of this thesis, it should be noted that CMCF problems can also be cast as non-linear mathematical programming problems. For example, the traffic equilibrium problem [11,33,48,53,54] can be formulated as a multi-commodity flow problem where:

i) the network arcs and nodes represent roadways and intersections, respectively; and

ii) the commodities represent specific origin to destination traffic flows.

In the traffic equilibrium problem, the bundle constraints are eliminated and instead, non-linearities in the objective function capture the effect on travel time of congestion in the network.

1.2 Literature Review

The linear CMCF problem is a **linear program** and can therefore be solved using a technique, developed in 1947 by George B. Dantzig [19], called the simplex method. Specialized solution techniques, however, are generally used. The most widely researched techniques are:

i) partitioning methods;

ii) resource-directive decomposition methods; and

iii) price-directive decomposition methods.

Assad [6] and Kennington [36] provide excellent CMCF surveys describing these solution techniques.

1.2.1 Partitioning Methods

To solve the CMCF problem, partitioning methods use the simplex procedure and partition the current basis to exploit the problem's underlying network structure. Grigoriadis and White [24,25] observe that only a small number of the bundle constraints are actually binding in practice. Thus, they include only a subset of the bundle constraints in their current basis and perform dual simplex steps using a generalization of Rosen's Primal Partition Programming [44]. Hartman and Lasdon [26], Kennington [34], Maier [38], and Saigal [45] propose compact inverse methods to solve the CMCF problem. These methods are specializations of Generalized Upper Bounding for block-angular systems [37].

1.2.2 Resource-Directive Decomposition

The resource-directive decomposition method is an iterative approach where arc capacities are repeatedly allocated among commodities until the optimal allocation is determined. For each given capacity allocation, the CMCF problem decomposes into single-commodity transshipment problems-- one for each commodity. (These single-commodity transshipment problems can be efficiently solved using the primal network simplex or the primal-dual out-of-kilter approaches.) If the current capacity allocation is not optimal, a new allocation is determined using either the method of tangential approximation [22,49]; the method of feasible directions [36]; or subgradient optimization [27,35].

1.2.3 Price-Directive Decomposition

Price-directive decomposition is driven by the economic principal that prices can be set such that supply and demand are in equilibrium. In the CMCF problem, the objective is to find for each arc, that price which achieves equilibrium between the amount of capacity demanded and that supplied.

The Dantzig-Wolfe decomposition method [18] is a price-directive solution approach which is said to have been inspired by the earlier work of Ford and Fulkerson [20] on the multi-commodity maximal flow problem. The Ford-Fulkerson price-directive decomposition approach was adapted and implemented for the multi-commodity minimal cost flow problem by Tomlin [52].

In Dantzig-Wolfe decomposition, a master problem (MP) and $|K|$ subproblems (SP^k) are repeatedly solved. Ford and Fulkerson [20] showed that the optimal CMCF flow assignment can be represented as a convex combination of single origin to destination chains for each commodity, i.e.:

$$z_{MP}^* = \text{MIN } \Sigma_{k\in K}\, \Sigma_{l\in Q^k}\, \Sigma_{ij\in A}\ \ c_{ij}{}^k b^k \delta_{ij}{}^{l,k} \lambda_l{}^k \qquad [1.5]$$

subject to:

$$\Sigma_{k\in K}\, \Sigma_{l\in Q^k}\, b^k \delta_{ij}{}^{l,k} \lambda_l{}^k + s_{ij} = d_{ij}, \quad \forall\ ij\in A;\ (-\pi_{ij}) \qquad [1.6]$$

$$\Sigma_{l\in Q^k}\, \lambda_l{}^k = 1, \quad \forall\ k\in K;\ \ (\sigma^k) \qquad [1.7]$$

$$\lambda_l{}^k \geq 0, \quad \forall\ k\in K,\ \forall\ l\in Q^k; \qquad [1.8]$$

where:

Q^k: set of **extreme points** (or equivalently, s^k to t^k

single chains) for the set $X^k = \{x_{ij}^k, \forall\ ij \in A: \Sigma_{j \in N} x_{ij}^k - \Sigma_{i \in N} x_{ij}^k = b^k$ if $i = s^k$, $= -b^k$ if $i = t^k$, and $= 0$ otherwise; $x_{ij}^k \geq 0, \forall\ ij \in A\}\ \forall\ k \in K$.

$\delta_{ij}^{l,k}$: $= 1$ if arc ij is included in the single chain $l \in Q^k$ for commodity $k \in K$, $= 0$ otherwise;

λ_l^k: decision variables (Note: the amount of flow on each extreme point chain $l \in Q^k$ is $b^k \lambda_l^k$, $\forall\ k \in K$).

Using the simplex method to solve MP, let $-\pi_{ij}$ and σ^k be the dual prices associated with constraints [1.6] and [1.7] respectively. The reduced costs of the variables λ_l^k and s_{ij} are then:

$$rc_l^k = \Sigma_{ij \in A}\, c_{ij}^k b^k \delta_{ij}^{l,k} + \Sigma_{ij \in A}\, \pi_{ij} b^k \delta_{ij}^{l,k} - \sigma^k, \quad \text{for } \lambda_l^k,\ \forall\ l \in Q^k,\ \forall\ k \in K. \qquad [1.9]$$

$$rc_{ij} = \pi_{ij} \quad \text{for } s_{ij},\ \forall\ ij \in A. \qquad [1.10]$$

Since MP is a minimization problem, any variable with negative reduced cost should be pivoted into the basis. Following along these lines, the variable with the most negative reduced cost should be pivoted into the basis. To determine the λ^k variable with the most negative reduced cost, consider the following subproblem, denoted SP^k:

$$z_{SP^k}{}^* = \text{MIN } \Sigma_{ij \in A}\, (c_{ij}^k + \pi_{ij}) x_{ij}^k \qquad [1.11]$$

subject to:

$$x_{ij}^k \in X^K. \qquad [1.12]$$

The optimal solution to SP^k (for each $k \in K$) is the assignment of all b^k units of flow to the minimum cost path from s^k to t^k. (This minimum cost path for k, denoted l^*, can be efficiently determined with a shortest path algorithm using revised arc costs of $c_{ij}^k + \pi_{ij}$.) The optimal objective function value for SP^k can therefore be written as:

$$z_{SP^k}^* = \Sigma_{ij\in A}\, c_{ij}^k b^k \delta_{ij}^{l*,k} + \Sigma_{ij\in A}\, \pi_{ij} b^k \delta_{ij}^{l*,k}, \ \forall\ k\in K \qquad [1.13]$$

and hence,

$$rc_{l*,}{}^{k} = z_{SP^k}^* - \sigma^k. \qquad [1.14]$$

If $z_{SP^k}^* < \sigma^k$, then for commodity k, there exists a candidate for entry into the basis. Thus, by locating the minimum cost chain, the variable with the most negative reduced cost is determined. Tomlin [52] therefore draws an analogy between Dantzig-Wolfe decomposition and decomposition of node-arc into arc-chain flows. Ford and Fulkerson [20] suggested that these shortest chains for entry into the basis be generated as needed. Using this idea, one implementation of the Dantzig-Wolfe price-directive decomposition algorithm is as follows:

Step 0: Initialization. Let the initial set of variables include:

i) slack variables s_{ij}, $\forall$ $ij\in A$; and

ii) artificial variables a^k, $\forall$ $k\in K$-- representing an infinite cost, infinite capacity, origin to destination, single arc path for commodity $k\in K$.

Step 1: Solve MP. Given the current set of variables, use the simplex method to solve MP and to obtain new dual prices.

Step 2: Dual Update. Update the cost on each arc ij as follows:

$$c'_{ij} = c_{ij} + \pi_{ij}, \quad \forall\ ij \in A. \qquad [1.15]$$

Then, solve SP^k for each $k \in K$.

Step 3: Test for Optimality. For all $k \in K$, if $z_{SP}k^* \geq \sigma^k$ STOP--the CMCF problem is optimally solved; otherwise continue. (Note: since artificial arcs with infinite capacity are included in the initial basis, the CMCF problem is always "feasible".)

Step 4: Basis Update. For all $k \in K$ with $z_{SP}k^* < \sigma^k$, add the variable (column) $\lambda_{1*}{}^k$ to MP. Go to Step 1.

Bazaraa and Jarvis [8], Chen and Dewald [15], Jarvis and Keith [30], Cremeans et. al. [17], Swoveland [49,50], Weigel and Cremeans [55] and Wollmer [58] report additional price-directive decomposition results for both the multi-commodity maximal flow problem and the multi-commodity minimum cost flow problem.

1.2.4 Other Solution Techniques

Jewell [31,32] developed a primal-dual simplex method to solve the multi-commodity maximal flow problem. The drawback to Jewell's approach is that it requires the identification of all forward and backward chains, as well as all loops for each commodity.

Gersht and Shulman [23] solve the linear CMCF problem using a barrier-penalty optimization algorithm specifically geared to the solution of CMCF problems with a large number of commodities.

1.3 Computational Experience

Table 1.1 summarizes the size of problems solved using the above discussed solution techniques. Test results of Swoveland [49] and Assad [5] independently showed that the Dantzig-Wolfe decomposition method is superior to the resource-directive decomposition method. Additionally, Assad and Swoveland both reported that Dantzig-Wolfe did not exhibit the tailing-off phenomenon generally observed. Tomlin [53], Swoveland [49,50], and Chen and Dewald [15] also reported success in solving CMCF problems using this decomposition strategy.

Computational experience with resource-directive techniques is provided by Held, Wolfe and Crowder [27] and Kennington and Shalaby [35]. They developed heuristics and used a subgradient approach to update capacity allocations. Assad [5] found that the subgradient approach converged for problems with 26 or less arcs but for problems with greater than 100 arcs or greater than 10 commodities, convergence was very slow. In fact, the solution time for the subgradient method was from 10 to 20 times slower than that for the Dantzig-Wolfe method.

Kennington [34,35] compared the performance of the resource-directive subgradient approach, the primal partitioning method and the simplex method for capacitated multi-commodity transportation problems. The subgradient approach was approximately 1.7 to 4 times faster than the partitioning approach, which was comparable to the APEX III [4] LP code.

NAME	SOLUTION METHOD	PROBLEM SIZE (n,m,mc,k)†
Tomlin [53]	Price-Directive Decomposition	(18,54,54,10-60) (50,152,152,63)
Swoveland [49,50]	Dantzig-Wolfe and Resource-Directive Decomposition	(356,1279,50-60,6) (92,224,50-60,4) (37,67,50-60,3)
Grigoriadis& White [24,25]	Primal Partitioning Algorithm	(8,15,1-15,5-40) (21,98,1-37,2-20)
Assad [5,7]	Decomposition	(47-83,100-270,100, 2-30)
Kennington [34,35]	Resource-Directive w/ Subgradient & Primal Partitioning	(8-12,8-12,8-12, 8-12)
		(#LP rows, #LP cols.)
Ali et al [3]	Primal-Partitioning & LP	(400-600,700-1000) (1395-2191,3165-21676)‡

†: n: number of nodes; m: number of arcs; mc: number of capacitated arcs; k: number of commodities.

‡: Real-world problems, the largest of which is solved on a CDC 6600 computer.

Table 1.1: Size of Problems Solved

Ali, et. al. [3] implemented the primal partitioning algorithm of Hartman and Lasdon [26] and solved three real-world problems ranging in size from 1395 to 2191 rows and from 3165 to 21676 columns. Their experience shows that the partitioning algorithm performed quite well on real-world problems where only a few of the coupling constraints are binding at optimality.

Additionally, Ali, et. al. generated a set of smaller test problems to compare the performance of primal-partitioning algorithms with the general LP codes MINOS [41], XMP [39], and LISS [1]. They found that in solving the CMCF problems, primal partitioning codes are from 2 to 3 times faster than the general LP codes.

Finally, Ho and Loute [28] tested their implementation of the Dantzig-Wolfe decomposition method against IBM's MPSX/370 LP software [29]. They performed computational tests for large-scale block-angular LP's with 411 to 5898 rows and 1411 to 12048 columns. In solving the set of block-angular problems, the Dantzig-Wolfe method was approximately 4 times slower than the general LP code. This result, which contradicts the computational findings reported above, may be explained by the fact that the set of test problems, although block-angular in form, are not CMCF problems specifically.

To conclude, in solving CMCF problems, computational testing shows that in general, price-directive decomposition methods outperform the partitioning method, the simplex method and resource-directive decomposition methods.

1.4 Thesis Outline

This chapter, which defines the CMCF problem and reviews existing techniques for its solution, is followed up by an in-depth review in

Chapter 2 of the primal-dual solution methodology. Then, Chapter 3 evaluates the solution methodologies presented in Chapters 1 and 2 and motivates the development of a new network-based, primal-dual algorithm (PDN) for the CMCF problem. Chapters 4, 5 and 6 describe the PDN algorithm in detail. Then, Chapters 7 and 8 present several algorithmic implementations for the CMCF problem and compare their performances. Finally, Chapter 9 summarizes the contributions of this thesis and describes directions of future CMCF research.

2. The Primal-Dual Method and the CMCF Problem

This chapter describes how the Capacitated Multi-Commodity Network Flow (CMCF) Problem can be solved using the primal-dual method. The theoretical details of the primal-dual method are reiterated here and are cast in terminology specific to the CMCF problem. A detailed understanding of the primal-dual method is essential in that it provides the foundation upon which follow-on chapters build.

Section 2.1 presents the path formulation of the CMCF problem. The path formulation is particularly useful to Section 2.2 which describes the theoretical underpinnings of the primal-dual method. Section 2.3 presents the Primal-Dual Algorithm itself and Section 2.4 shows that the Primal-Dual method optimally solves the CMCF problem in a finite number of steps. Finally, Section 2.5 summarizes the basic idea of the primal-dual method.

2.1 Path Formulation of the CMCF Problem

The path formulation of the CMCF problem in standard form, P, is as follows:

$$z^*_P = \text{MIN } \Sigma_{k\in K} \Sigma_{p\in P^k} c_p{}^k x_p{}^k \qquad [2.1]$$

subject to:

$$\Sigma_{k\in K} \Sigma_{p\in P^k} x_p{}^k \delta_{ij}{}^{p,k} + s_{ij} = d_{ij}, \quad \forall\ ij\in A \qquad [2.2]$$

$$\Sigma_{p\in P^k} x_p{}^k = b^k, \quad \forall\ k\in K \qquad [2.3]$$

$$x_p{}^k \geq 0,\ \forall\ p\in P^k,\ \forall\ k\in K;\ s_{ij} \geq 0, \quad \forall\ ij\in A. \qquad [2.4]$$

where:

K: set of $1,\ldots,|K|$ commodities, where each origin to destination demand is considered a commodity;

P^k: set of all distinct, simple, directed paths from the origin to the destination of commodity k, $\forall\ k \in K$;

$\delta_{ij}^{p,k}$: $= 1$ if arc $ij \in A$ is contained on path $p \in P^k$, $k \in K$; and $= 0$ otherwise;

c_p^k: cost of path $p \in P^k$, $k \in K$ where:

$$c_p^k = \Sigma_{ij \in A}\ c_{ij}^k \delta_{ij}^{p,k};$$

x_p^k: flow of commodity $k \in K$ on path $p \in P^k$;

b^k: quantity demanded/supplied of commodity $k \in K$;

d_{ij}: capacity of arc $ij \in A$;

s_{ij}: slack variables for constraints [2.2]-- representing the amount of unused arc capacity, i.e.:

$$s_{ij} = d_{ij} - X_{ij};\ \text{where } X_{ij} = \Sigma_{p \in P^k}\ \Sigma_{k \in K}\ x_p^k \delta_{ij}^{p,k};$$

Equations [2.2] limit the total flow on each arc to the capacity of that arc. Equations [2.3] are the demand constraints which ensure that for each commodity, the total flow on all paths from the origin to the destination exactly equals supply. Finally, equations [2.4] are the flow and slack non-negativity constraints. The objective, equation [2.1], is to minimize the total cost of flow on all paths.

The dual of the CMCF problem, D, can be correspondingly formulated as follows:

$$z^*_D = \text{MAX}\ \Sigma_{ij \in A}\ -\pi_{ij} d_{ij} + \Sigma_{k \in K}\ \sigma^k b^k \qquad [2.5]$$

subject to:

$$-\Sigma_{ij\in A}\ \pi_{ij}\delta_{ij}^{p,k} + \sigma^k \le c_p^k, \quad \forall\ p\in P^k,\ \forall\ k\in K \qquad [2.6]$$

$$\pi_{ij}\ge 0, \qquad \forall\ ij\in A \qquad [2.7]$$

σ^k unrestricted in sign (u.i.s.), $\forall$ k$\in$K. [2.8]

where:

π_{ij}: negative of the dual variables for constraints [2.2]--can be interpreted as the "price" on arc ij$\in$A;

σ^k: dual variables on constraints [2.3]-- can be interpreted as the "cost" or "length" of the shortest path for commodity k$\in$K. (The terms path length and path cost are used interchangeably here.) The length of path p for commodity k, λ_p^k, is defined as:

$$\lambda_p^k = \Sigma_{ij\in A}\ (c_{ij}^k + \pi_{ij})\delta_{ij}^{p,k}. \qquad [2.9]$$

Given the above interpretation of the dual variables, equations [2.6] require for each commodity, that all the origin-to-destination paths are at least as long as the shortest path. Equations [2.7] restrict the dual arc prices to non-negative values and the objective, equation [2.5], can be loosely interpreted as balancing the conflicting goals of maximizing each commodity's shortest path length and minimizing dual arc prices.

The objective of the CMCF problem, in light of the above path formulation, is to find a minimum-cost assignment of flow to paths such that all commodity-specific demand is satisfied and no arcs are over-capacitated. This interpretation of the CMCF problem facilitates the description of the primal-dual method.

2.2 Primal-Dual Method

The primal-dual method provides the theoretical foundation of the work described in this thesis. Thus, to obtain a thorough understanding of the methodology, the results of the primal-dual method are reiterated here and interpreted specifically with respect to the CMCF problem.

The primal-dual method, like dual methods, always maintains dual feasibility. It begins with a dual feasible solution and then, given this dual solution, tries to construct a primal solution which obeys complementary slackness and feasibility conditions. If such a solution does not exist, a different dual feasible solution is determined and another search for a satisfactory primal solution is launched. The primal-dual method iterates in this manner until, for some given dual feasible solution, a primal solution obeying complementary slackness and primal feasibility is found. Once all of these conditions are satisfied, basic mathematical programming theory shows that optimality is achieved.

The Complementary Slackness conditions for the CMCF problem represent a theoretical tie between the primal and dual solutions. These conditions are as follows:

CS#1: $-\pi_{ij}(\Sigma_{k\in K}\ \Sigma_{p\in P^k}\ x_p^k\delta_{ij}^{p,k} + s_{ij} - d_{ij}) = 0,\ \forall ij\in A;$ [2.10]

CS#2: $\sigma^k(\Sigma_{p\in P^k}\ x_p^k - b^k) = 0,\qquad \forall\ k\in K;$ [2.11]

CS#3: $x_p^k(c_p^k + \Sigma_{ij\in A}\ \pi_{ij}\delta_{ij}^{p,k} - \sigma^k) = 0,\ \forall p\in P^k,\ \forall k\in K;$ [2.12]

CS#4: $\pi_{ij}(s_{ij}) = 0,\qquad \forall\ ij\in A.$ [2.13]

In P, equations [2.2] and [2.3] are equality constraints and thus, complementary slackness conditions #1 and #2 are satisfied for any primal feasible solution. Complementary slackness conditions #3 and #4, however, are not so trivially satisfied.

CS #3 conditions restrict the assignment of flow to least cost paths; where path cost, equation [2.9], is a function of both a **primal flow cost** and a **dual arc price**. CS #4 conditions restrict the value of slack to zero for each arc with a non-zero dual price. Said another way, CS #4 conditions require that total flow exactly equals capacity on each arc with a non-zero dual price. These complementary slackness conditions highlight the "push-pull" nature of the dual arc prices and primal flows. They show that arc prices restrict flow assignments and flow assignments restrict arc prices. Highlighting this effect, the problem can be viewed as finding a set of feasible dual prices and feasible primal flows which are in equilibrium with respect to the complementary slackness conditions.

From equations [2.12] and [2.13], CS #3 and CS #4 conditions are satisfied if flow is restricted only to least cost paths and the value of slack is positive only on arcs with dual price equal to zero. Let

J denote the set containing least cost paths and slack variables which can take on non-zero values. Using the terminology of Papadimitriou and Steiglitz [42], the paths contained within set J are "admissible." In mathematical terms, all paths (and all arcs contained in these paths) which satisfy the following relationship are admissible:

$$-\Sigma_{ij\in A}\ \pi_{ij}\delta_{ij}^{p,k} + \sigma^k = c_p^k, \quad p\in P^k,\ k\in K; \qquad [2.14]$$

Similarly, slack variable s_{ij} is admissible if arc ij satisfies the following criterion:

$$\pi_{ij} = 0, \quad ij\in A. \qquad [2.15]$$

Then, any primal feasible solution to P which exclusively utilizes the elements of set J (meaning $x_p^k>0$ only if $p\in J$ and $s_{ij}>0$ only if $s_{ij}\in J$), is a solution which satisfies all complementary slackness conditions. This leads to the first basic primal-dual result:

Result #1: Any feasible solution to P exclusively using the elements of set J (defined for a given feasible solution in D), is an optimal solution to P.

Considering the CMCF problem specifically, this result can be explained as follows:

Proof: A feasible solution to P satisfies primal constraints [2.2] and [2.3], thereby satisfying CS #1 and CS #2 conditions. Since the feasible solution exclusively uses the elements of set J, flow is assigned only to shortest paths (equation [2.14]), and the value of slack is non-zero only for arcs with dual arc price equal to zero (equation [2.15]). Hence, CS #3 and CS #4 conditions are satisfied.

For a given dual feasible solution, any feasible solution to P exclusively using the elements of set J, satisfies primal feasibility and complementary slackness conditions. Therefore, optimality in P is achieved. ∎

Thus, the set J is defined such that if **feasibility** is achieved for P, **optimally** is simultaneously achieved.

Figure 2.1 shows how the primal-dual method implements these ideas in an iterative framework. In Step 0, an initial feasible solution in D is generated. Step 1 uses the current dual solution to define the set of admissible elements. Then Step 2, using only these admissible elements, assigns flow with the objective of achieving feasibility in P. (This problem is referred to as the Restricted Primal (RP) problem.) If feasibility in P is achieved using only admissible elements, then optimality in P is simultaneously achieved (Result #1). It is possible that the admissible set is too restrictive, however, and no feasible solution to P can be found using only the elements of J. In this case, Step 3 gauges the closeness of the solution to optimality by computing an infeasibility measure, denoted z^*_{RP}. As shown later, optimality in P is achieved when $z^*_{RP} = 0$. If optimality is achieved, the algorithm terminates; otherwise another, less restrictive, dual feasible solution for D is generated in Step 4 and starting with Step 1, the entire process repeats.

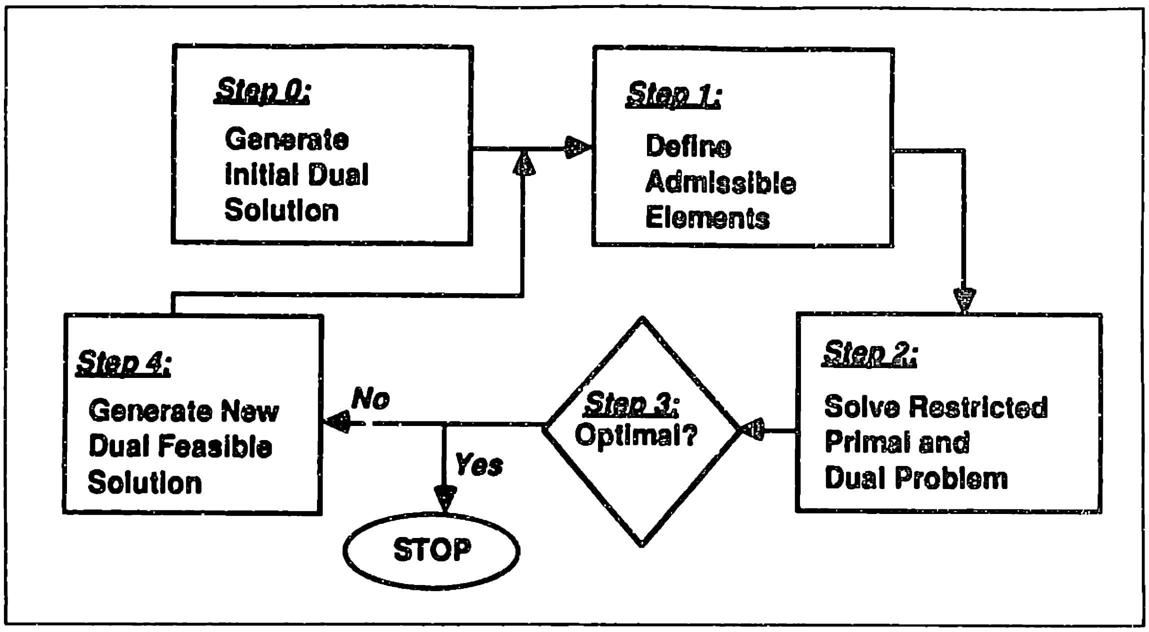
Step 0:
Generate Initial Dual Solution
Step 1:
Define Admissible Elements
Step 2:
Solve Restricted Primal and Dual Problem
Step 3:
Optimal?
No
Yes
Step 4:
Generate New Dual Feasible Solution
STOP

Figure 2.1: Primal-Dual Method

The following sections, 2.2.1 through 2.2.5, describe in further detail the steps of the primal-dual method.

2.2.1 Step 0: Generate Initial Solution

An initial dual feasible solution to the CMCF problem can be easily achieved by:

i) setting all dual arc prices to zero; and

ii) determining commodity-specific dual node prices with a shortest path algorithm.

2.2.2 Step 1: Define Admissible Elements

For a given current dual feasible solution, admissible paths are (least cost) paths satisfying equation [2.14] and admissible slacks are slack variables associated with arcs satisfying equation [2.15].

2.2.3 Step 2: Solve the Restricted Primal and Dual Problem

The general framework of the primal-dual method, as presented in Papadimitriou and Steiglitz [42] and in Shapiro [47], is adopted and applied here to the CMCF problem specifically. The RP problem can be formulated in several different ways depending on the variables introduced. The formulation adopted here is as follows:

$$\text{MIN } z_{RP} = \Sigma_{ij\in A} (w_{ij}^- + w_{ij}^+) \qquad [2.16]$$

subject to:

$$\Sigma_{k\in K} \Sigma_{p\in J^k} x_p^k \delta_{ij}^{p,k} + w_{ij}^+ - w_{ij}^- = d_{ij}, \quad \forall\ ij\in IJ^+(\pi); \qquad [2.17a]$$

$$\Sigma_{k\in K} \Sigma_{p\in J^k} x_p^k \delta_{ij}^{p,k} + s_{ij} - w_{ij}^- = d_{ij}, \quad \forall\ ij\in IJ^+(\pi)^c; \qquad [2.17b]$$

$$\Sigma_{p\in J^k} x_p^k = b^k, \quad \forall\ k\in K; \qquad [2.18]$$

$$x_p^k \geq 0,\ \forall\ p\in J^k,\ \forall\ k\in K;\ x_p^k = 0,\ \forall\ p\in (J^k)^c,\ \forall\ k\in K; \qquad [2.19]$$

$$w_{ij}^-,\ w_{ij}^+ \geq 0,\ \ \forall\ ij\in A;\ \ w_{ij}^+ = 0,\ \forall\ ij\in IJ^+(\pi)^c. \qquad [2.20a]$$

$$s_{ij} \geq 0,\ \forall\ ij\in A;\ s_{ij} = 0,\ \forall\ ij\in IJ^+(\pi). \qquad [2.20b]$$

where:

J^k:	set of least-cost (shortest) paths in set J for commodity $k\in K$;
$(J^k)^c$:	complement of set J^k;
$IJ^+(\pi)$:	set of arcs $ij\in A$ with $\pi_{ij} > 0$;
$IJ^+(\pi)^c$:	complement of set $IJ^+(\pi)$;

w_{ij}^{+}, w_{ij}^{-}: artificial variables measuring infeasibility of the flow assignment on each arc $ij \in A$.

The objective of the RP problem is to find, using only admissible elements, the flow assignment which minimizes the number of violations to feasibility in P. Section 2.2.4 shows that when the objective function value of RP equals zero, primal feasibility and optimality in P are simultaneously achieved.

There are two major differences between the RP problem (equation [2.16] through [2.20]) and the CMCF problem, P (equations [2.1] through [2.4]). First, the objective function of RP, unlike the objective function of P, does not explicitly deal with arc costs. Instead, arc costs are implicitly considered by restricting the assignment of flow to admissible paths only. The need to evaluate a potentially complicated cost function is therefore alleviated and the objective in the RP problem is reduced to finding a feasible solution only. Second, the concept of admissibility reduces the size of the problem by eliminating from consideration all paths which are not shortest and all slack variables for arcs with non-zero dual prices. As a result, the RP problem is much smaller than P.

The dual of the RP problem, DRP, is as follows:

$$\text{MAX } z_{DRP} = \Sigma_{ij \in A} -\Delta\pi_{ij} d_{ij} + \Sigma_{k \in K} \Delta\sigma^{k} b^{k} \qquad [2.21]$$

subject to:

$$-\Sigma_{ij \in A} \Delta\pi_{ij} \delta_{ij}^{p,k} + \Delta\sigma^{k} \leq 0, \quad \forall\ p \in J^{k}, \quad \forall\ k \in K; \qquad [2.22]$$

$-1 \leq \Delta\pi_{ij} \leq 1, \qquad \forall\ ij \in IJ^+(\pi);$ [2.23]

$0 \leq \Delta\pi_{ij} \leq 1, \qquad \forall\ ij \in IJ^+(\pi)^c;$ [2.24]

$\Delta\sigma^k$ u.i.s., $\qquad \forall\ k \in K.$ [2.25]

where:

$\Delta\pi_{ij}$: dual arc price on arc ij: prices associated with the bundle constraints (equations [2.17])-- can be interpreted as the "direction" or "relative magnitude" of change for the dual arc prices in D;

$\Delta\sigma^k$: dual prices associated with the demand constraints (equations [2.18])-- can be interpreted as the "direction" or "relative magnitude" of change for the dual price at the destination node for commodity k.

Certain observations can be made here to provide a better understanding of the restricted dual problem. First, using the simplex method, the optimal solution to the RP problem is a basis containing flow-carrying paths for each commodity. The reduced cost of each basic path p for commodity k, $c'_p{}^k$, is equal to zero:

$$c'_p{}^k = 0 - [-\Sigma_{ij \in A}\ \Delta\pi_{ij}\delta_{ij}^{p,k} + \Delta\sigma^k] = 0; \qquad \forall\ p \in J^k \ \&\ \text{basic}, \quad \forall\ k \in K \qquad [2.26]$$

or, rewriting:

$$\Sigma_{ij \in A}\ \Delta\pi_{ij}\delta_{ij}^{p,k} = \Delta\sigma^k. \quad \forall\ p \in J^k \ \&\ \text{basic}, \quad \forall\ k \in K. \qquad [2.27]$$

For now, interpret $\Delta\sigma^k$ as the change in the length of the shortest path for commodity $k \in K$, and interpret $\Delta\pi_{ij}$ as the change in the dual price on arc $ij \in A$. (Each of these values will be multiplied by some factor to achieve the actual changes in shortest path lengths and dual arc prices.) Then, equations [2.27] show that every basic path for each commodity k is equally altered in length by an amount equal to $\Delta\sigma^k$. Furthermore, for each basic path for commodity k, $\Delta\sigma^k$ is equal to the sum of the dual price adjustments on all arcs contained in the path.

Since only basic paths contain flow, the dual objective function to the DRP problem, equation [2.21], can be rewritten using equations [2.18] and [2.27]:

$$z^*_{DRP} = \text{MAX } \Sigma_{ij \in A} -\Delta\pi_{ij} d_{ij} + \Sigma_{k \in K} \Sigma_{p \in J^k} \Sigma_{ij \in A} \Delta\pi_{ij} \delta_{ij}{}^{p,k} x_p{}^k. \qquad [2.28]$$

To simplify, recall that by definition, total flow is:

$$X_{ij} = \Sigma_{k \in K} \Sigma_{p \in J^k} \delta_{ij}{}^{p,k} x_p{}^k, \quad \forall\ ij \in A. \qquad [2.29]$$

Substituting equation [2.29] into equation [2.28], the dual objective function becomes:

$$z^*_{DRP} = \text{MAX } \Sigma_{ij \in A} \Delta\pi_{ij} [X_{ij} - d_{ij}]. \qquad [2.30]$$

These manipulations show the direct effect of alterations in dual prices on the objective function of the DRP problem. It is clear that equation [2.30] is maximized if:

i) a positive adjustment is made to π_{ij} for arcs with total flow exceeding capacity, i.e.

$$\Delta\pi_{ij} > 0 \text{ if } X_{ij}-d_{ij} \geq 0, \quad ij \in A; \text{ and} \qquad [2.31a]$$

ii) a negative adjustment is made to π_{ij} for arcs with capacity exceeding total flow and with positive dual price, i.e.

$$\Delta\pi_{ij} < 0 \text{ if } X_{ij}-d_{ij} \leq 0 \text{ and } \pi_{ij} > 0, \; ij \in A. \qquad [2.31b]$$

This suggests an economic interpretation as follows. If an arc has flow exceeding capacity, then demand for its resource, namely capacity, exceeds supply and thus, its price should be increased. Likewise, an arc with a positive dual price and flow strictly less than capacity has its resource priced above the market price and for equilibrium to exist, the arc's dual price should be decreased.

From the above analysis, the DRP problem can be interpreted as **maximally** increasing dual prices on arcs with flow above capacity and **maximally** decreasing (subject to the non-negativity restriction on dual arc prices) dual prices on arcs with flow below capacity such that all paths containing flow of a commodity are equally altered in length.

2.2.4 Step 3: Test for Optimality

A basic result of the primal-dual method is that optimality is achieved in P when the following occurs:

<u>Result #2:</u> The CMCF problem, P, is optimally solved if the optimal objective function value of the RP problem, z^*_{RP}, is equal to zero.

This primal-dual result can be proven for the CMCF RP problem formulation (equations [2.21] - [2.25]) as follows:

<u>Proof:</u> From equation [2.16]:

$$z^*_{RP} = \Sigma_{ij \in A} (w_{ij}^- + w_{ij}^+)$$

where, from equation [2.20], w_{ij}^+, $w_{ij}^- \geq 0$, $\forall$ $ij \in A$.

If $z^*_{RP} = 0$, it follows that w_{ij}^+, $w_{ij}^- = 0$, $\forall$ $ij \in A$. From equation [2.17a], w_{ij}^+, $w_{ij}^- = 0$, $\forall ij \in A$ implies that all arcs $ij \in A$ with $\pi_{ij} > 0$ have total flow exactly equal to capacity. Furthermore, from equations [2.17b] and [2.19], w_{ij}^+, $w_{ij}^- = 0$, $\forall ij \in A$ imply that all arcs $ij \in A$ with $\pi_{ij} = 0$ have total flow not greater than capacity. Hence, the primal solution in RP is feasible in P. Furthermore, by design, the optimal solution in RP exclusively utilizes the elements of set J. Hence, by Result #1, optimality in P is achieved. ■

If z^*_{RP} is strictly greater than zero, then $w_{ij}^- > 0$ for some $ij \in A$ and/or $w_{ij}^+ > 0$ for some $ij \in A$. If $w_{ij}^- > 0$ for an arc $ij \in A$, then it can be seen from equations [2.16] and [2.17] that total flow must exceed capacity for that arc ij, thereby violating feasibility (equations [2.2]) in P. If $w_{ij}^+ > 0$ for an arc $ij \in A$, then, again from equations [2.16] and [2.17], total flow plus slack must be strictly less than capacity for that arc $ij \in IJ^+(\pi)$, thereby again violating feasibility (equations [2.2]) in P. Thus, if z^*_{RP} is strictly greater than zero, feasibility and hence, optimality in the CMCF problem is not achieved.

2.2.5 Step 4: Generate a Feasible Solution for D

Section 2.2.4 showed that if the optimal objective function value of the RP problem is strictly greater than zero, neither feasibility nor optimality in P is achieved for the current dual solution. Consequently, a new set of feasible prices in D must be generated, and another iteration of the primal-dual method is necessary. The new dual prices are determined as follows:

$$\pi'_{ij} = \pi_{ij} + \theta_i \Delta\pi_{ij}, \quad ij \in A; \qquad \text{and}$$
$$\sigma^{k'} = \sigma^k + \theta_i \Delta\sigma^k, \quad k \in K. \qquad [2.32]$$

where:

π_{ij}, σ^k: dual prices for D, used in the ith iteration of the primal-dual algorithm;

π'_{ij}, $\sigma^{k'}$: revised dual prices for D, used in the i+1st iteration of the primal-dual algorithm;

$\Delta\pi_{ij}$, $\Delta\sigma^k$: optimal prices for the ith DRP problem, DRP_i;

θ_i: multiplier of optimal prices for the DRP_i problem.

Thus, at the ith iteration of the primal-dual method, the optimal solution to the DRP problem provides the direction in which to alter the dual prices in D. The magnitude of the alteration is measured by θ_i, where θ_i is restricted in size by the following requirements:

i) the altered dual prices should be feasible in D;

ii) the altered dual prices should maintain admissibility for all basic elements in the optimal solution to RP_i; and

iii) the altered dual prices should not decrease the objective function value in D.

The following sections describe the effects of these requirements on the definition of θ_i.

<u>**2.2.5.1 Dual Price Adjustments, Admissibility and Dual Feasibility**</u>

As stated previously, each dual price adjustment must not only produce dual feasible prices but must also produce new dual prices which satisfy the admissibility conditions (equations [2.14] and [2.15]) for all elements in the optimal basis of the ith RP problem. The implications of these requirements are captured in the following primal-dual result and presented here for the CMCF problem in particular.

<u>**Result #3:**</u> To satisfy the requirements that the adjusted dual prices maintain feasibility in D and admissibility for all basic elements in the optimal solution to RP_i, θ_i is defined as follows:

$$\theta_i = \mathrm{MIN}\,[(\mathrm{MIN}_\alpha(c_p{}^k + \Sigma_{ij\in A}\, \pi_{ij}\delta_{ij}{}^{p,k} - \sigma^k)/ \\ (-\Sigma_{ij\in A}\, \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k));\ \mathrm{MIN}_\beta(-\pi_{ij}/\Delta\pi_{ij})] \qquad [2.33]$$

where:

$\alpha = \{p|p\in(P^k),\ \forall k\in K,\ -\Sigma_{ij\in A}\, \Delta\pi_{ij}\delta_{ij}{}^{pi,k} + \Delta\sigma^k - \gamma^{pk} > 0\}$;

$\beta = \{ij|ij\in A\ \&\ \Delta\pi_{ij}<0\}$;

<u>Proof:</u> For the adjusted dual prices to maintain dual feasibility, each adjusted dual arc price must be non-negative, i.e.:

$$\pi_{ij} + \theta_i \Delta\pi_{ij} \geq 0, \quad \forall\ ij \in A. \qquad [2.34]$$

Rewriting equation [2.34]:

$$\theta_i \leq -\pi_{ij}/\Delta\pi_{ij}, \qquad \forall\ ij \in A \text{ with } \Delta\pi_{ij} < 0, \qquad [2.35]$$

or, equivalently:

$$\theta_i = \text{MIN}_\beta(-\pi_{ij}/\Delta\pi_{ij}), \quad \text{where } \beta = \{ij \mid ij \in A \cap \Delta\pi_{ij} < 0\} \qquad [2.36]$$

θ_i must be bounded not only to maintain non-negativity of each dual arc price as in equation [2.36], but also to ensure that the adjusted dual prices satisfy dual feasibility constraints [2.6]:

$$-\Sigma_{ij \in A}\ (\pi_{ij} + \theta_i \Delta\pi_{ij})\delta_{ij}{}^{p,k} + \sigma^k + \theta_i \Delta\sigma^k \leq c_p{}^k,$$
$$\forall p \in P^k,\ \forall k \in K. \qquad [2.37]$$

For now assume (it will be shown later) that θ_i is non-negative. Then from equation [2.37], since the current set of dual prices are feasible in D, the adjusted dual prices can violate feasibility constraints [2.6] only if:

$$-\Sigma_{ij \in A}\ \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k = \gamma_p{}^k > 0,\ \forall\ p \in P^k,\ \forall\ k \in K. \qquad [2.38]$$

Thus, solving equation [2.37] for θ_i and using the observation expressed in equation [2.38], the adjusted dual prices are guaranteed to satisfy dual feasibility constraints [2.6] if θ_i is defined as follows:

$$\theta_i \leq (c_p{}^k + \Sigma_{ij\in A}\ \pi_{ij}\delta_{ij}{}^{p,k} - \sigma^k)/(-\Sigma_{ij\in A}\ \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k),$$
$$-\Sigma_{ij\in A}\ \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k = \gamma_p{}^k > 0,\ \forall\ p\in P^k,\quad \forall\ k\in K. \qquad [2.39]$$

Equations [2.36] and [2.39] define θ_i such that the adjusted dual prices are feasible in D.

Consider now, the requirement that the basic elements in the optimal solution to the RP problem remain admissible after the dual price adjustment. Since admissibility is defined both for paths (equation [2.14]) and for slack variables (equation [2.15]), consider the two cases separately.

Take first the case of a slack variable, s_{ij}, in the optimal basis for the RP problem. Since s_{ij} is included in the RP formulation, from admissibility it follows that $\pi_{ij} = 0$. Furthermore, from basic mathematical programming theory, the reduced cost of this basic s_{ij} variable in the optimal solution to the RP problem is:

$$c'_{ij} = 0 - [-\Delta\pi_{ij}] = 0,\quad \forall\ ij \text{ such that } s_{ij} \text{ basic.} \qquad [2.40]$$

Equation [2.40] shows that $\Delta\pi_{ij} = 0$ for all arcs $ij\in A$ with s_{ij} basic. Hence, for each arc $ij\in A$ with s_{ij} basic, the adjusted dual prices, $\pi_{ij} + \theta_i\Delta\pi_{ij}$, remain equal to 0 for any θ_i. Thus, all basic slack variables remain admissible with the altered dual prices.

Now consider the final requirement that all flow carrying paths remain admissible with the adjusted dual prices. The reduced cost value of each flow-carrying (basic) path in the optimal solution to the RP problem is:

$$c'_p{}^k = 0 - [-\Sigma_{ij\in A}\ \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k] = 0,$$
$$\forall\ p\in J^k \cap p \text{ basic},\ k\in K. \qquad [2.41]$$

Plugging the adjusted dual prices (as defined in equations [2.32]) into admissibility condition [2.14], and using the relationship in equation [2.41], it follows that for any θ_i:

$$-\Sigma_{ij\in A}\ (\pi_{ij} + \theta_i\Delta\pi_{ij})\delta_{ij}{}^{p,k} + \sigma^k + \theta_i\Delta\sigma^k = c_p{}^k,$$
$$\forall\ p\in J^k \cap p \text{ basic},\ k\in K. \qquad [2.42]$$

Hence, all basic admissible paths remain admissible after dual prices are adjusted in accordance with definition [2.32].

Thus, the adjusted dual prices assure **feasibility in D** and **continued admissibility of all basic elements in the optimal solution to the RP problem** as long as θ_i satisfies equations [2.36] and [2.39]. Equation [2.36] together with equations [2.39] form the definition of θ_i as presented in equation [2.33]. ■

Note that the numerator of the first term in equation [2.33] represents the difference between the length of path p and the shortest path length for commodity k∈K and thus, is non-negative. The denominator of this term is strictly positive by definition. Thus, the first term of equation [2.33] is always non-negative. Likewise, the second term of equation [2.33] is always non-negative because, by dual feasibility, $-\pi_{ij}$ is less than 0 and by the definition of β, $\Delta\pi_{ij}$ is less than 0. Therefore, it follows that Θ_i is always non-negative.

Papadimitriou and Steiglitz [42] prove the following result for the primal-dual methodology:

Result #4: The definition of Θ_i in equation [2.33] ensures that if there is a feasible solution to P, at least one inadmissible element becomes admissible with the adjusted dual prices.

Following is the interpretation and proof of this result for the CMCF problem specifically:

Proof: If P is feasible, Θ_i is determined either by the first term or the second term in equation [2.33]. If the first term determines Θ_i, there is some path $p^*\epsilon\alpha$ which, by rewriting equation [2.33], satisfies:

$$-\Sigma_{ij\epsilon A}\ (\pi_{ij} + \Theta_i\pi_{ij})\delta_{ij}{}^{p^*,k^*} + \sigma^{k^*} + \Theta_i\Delta\sigma^{k^*} = c_{p^*}{}^{k^*},$$
$$\gamma_{p^*}{}^{k^*} > 0,\ p^*\epsilon P^{k^*}. \qquad [2.43]$$

Thus, given the adjusted dual prices, p^* is an admissible path (equation [2.14]).

Using equation [2.41] and recalling that RP is a minimization problem, it follows that all admissible paths in J have the following reduced costs:

$$c'_p{}^k = 0 - (-\Sigma_{ij\epsilon A}\ \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k) = -\gamma_p{}^k \geq 0. \qquad [2.44]$$

(This inequality also follows from dual feasibility requirements in the DRP problem.)

Equations [2.43] and [2.44] show that prior to the dual price adjustment, $\gamma_{p*}{}^{k*} > 0$ and thus, p^* is an inadmissible path. After the dual price adjustment, however, p^* is an admissible path. Thus, if θ_i is determined by the first term in equation [2.33], at least one inadmissible path becomes admissible.

A similar argument follows when θ_i is determined by the second term in equation [2.33]. In this case, there is some $ij^* \in \beta$ which satisfies:

$$-(\pi_{ij*} + \theta_i \Delta\pi_{ij*}) = 0, \quad ij* \in \beta. \qquad [2.45]$$

Thus, the adjusted dual price for arc ij^* is reduced to 0 and s_{ij*} becomes admissible by equation [2.15].

Using equation [2.40] and again recalling that RP is a minimization problem, the reduced cost of each admissible s_{ij} variable is:

$$c'_{ij} = 0 - (-\Delta\pi_{ij}) \geq 0. \qquad [2.46]$$

(This inequality also follows from the dual feasibility requirements in the DRP problem.)

Since $ij^* \in \beta$, $\Delta\pi_{ij} < 0$, and thus equation [2.46] shows that prior to the dual price adjustment, s_{ij*} is inadmissible. However, equation [2.45] shows that after the dual price adjustment, s_{ij*} is admissible. Thus, if θ_i is determined by the second term in equation [2.33], at least one inadmissible slack variable becomes admissible. ■

2.2.5.2 Dual Price Adjustments and the Dual Objective Function Value

In primal-dual methods, Papadimitriou and Steiglitz [42] show that ascent of the objective function in D occurs when the dual prices are altered and a new dual feasible solution is generated (Step 4). This result holds when the primal-dual method is used to solve the CMCF problem as follows:

Result #5: If dual prices are altered by a non-zero amount in accordance with equations [2.32] and [2.33], the dual objective function in D will increase.

Proof: Substituting the revised dual prices into equation [2.5], the new dual objective function value for D is:

$$z'^{*}_{D} = \Sigma_{ij\in A} -\pi'_{ij}d_{ij} + \Sigma_{k\in K} \sigma^{k\prime}b^{k}. \qquad [2.47]$$

Rewriting equation [2.47] using equation [2.32]:

$$z'^{*}_{D} = \Sigma_{ij\in A} -\pi_{ij}d_{ij} + \Sigma_{k\in K} \sigma^{k}b^{k} + \Sigma_{ij\in A} -\theta_{i}\Delta\pi_{ij}d_{ij} + \Sigma_{k\in K} \theta_{i}\Delta\sigma^{k}b^{k}. \qquad [2.48]$$

and simplifying:

$$z'^{*}_{D} = z^{*}_{D} + \theta_{i}(z^{*}_{DRP}). \qquad [2.49]$$

Section 2.2.4 showed that until optimality in P is achieved, the optimal objective function of the ith RP problem is strictly positive. Since the optimal objective function values of the RP_i and DRP_i

problems are equal, it follows that the optimal objective function value for the DRP_i problem is positive (at each iteration i) until optimality is achieved. Furthermore, Section 2.2.5.1 showed that θ_i is always non-negative. In fact, if dual prices are altered by a non-zero amount, then from equation [2.32], θ_i must be strictly positive. Combining these observations together with equation [2.49], it follows that for a non-zero dual price adjustment:

$$z'^*_D > z^*_D. \qquad [2.50]$$

$\blacksquare$

If all paths are admissible and the $\Delta\pi_{ij}$ values for all arcs $ij \in A$ are non-negative, then $\gamma_p{}^k \le 0$ for all paths (see equation [2.44]) and a dual price adjustment using any $\theta_i \ge 0$ does not threaten dual feasibility or admissibility requirements (Result #3). In other words, the dual prices can be altered by an unbounded θ_i amount. An unbounded increase in θ_i results in an unbounded dual objective function value (Result #5) and hence, the primal problem P is infeasible.

2.3 Primal-Dual Algorithm-- A Summary

To summarize, the primal-dual method begins with a feasible solution to the dual of the CMCF problem. This dual solution is used to define a set of admissible paths and slack variables to which flow may be assigned. The restricted primal problem, RP, is to find the flow assignment, using exclusively these admissible elements, whose value,

z^*_{RP}, is equal to 0. If there is no such flow assignment, the current dual solution is not optimal for the CMCF problem. Thus, a new feasible solution for D is determined (using the solution to the DRP problem), and the entire primal-dual process repeats. As shown in Figure 2.2, the primal-dual method iterates in this manner until optimality in P is achieved.

2.4 Finiteness

At each iteration of the primal-dual method, the RP problem can be solved using the simplex algorithm. As a result, Papadimitriou and Steiglitz [42] as well as Shapiro [47] show the primal-dual results previously presented, the primal-dual method optimally solves any standard LP problem. This results applies specifically to the CMCF problem as follows:

Result #6: If P is feasible and $z^*_{RP} > 0$, then (using standard perturbation techniques to eliminate degeneracy), $z^*_{RPi+1} < z^*_{RPi}$ where:

z^*_{RPi}, z^*_{RPi+1}: optimal objective function value for the ith and the i+1st RP problems, respectively.

Proof: From Result #3, all basic elements in the optimal solution to the ith RP problem remain admissible given the newly adjusted dual prices. Thus, the **optimal solution** to the ith RP problem is a **feasible starting solution** for the i+1st RP problem and, it follows that z^*_{RPi} provides an upper bound on the value of z^*_{RPi+1}, i.e. $z^*_{RPi+1} \leq z^*_{RPi}$.

```
Program Primal-Dual Method

Given:  $\pi$, $\sigma$ dual feasible

infeas ← .false.
opt ← .false.

DO WHILE (.not.opt .and. .not. infeas)  !Until optimal

     CALL DEFINE_SET_J                 !Finds admissible paths
     CALL SOLVE_RP                     !Solves RP with SIMPLEX

     IF ($z^*_{RP} = 0$) THEN              !Optimality test
         opt ← .true.
     ELSE IF ($\gamma^k<0, \forall p\in J^k, \forall k\in K$ .and. $\Delta\pi_{ij}\geq 0, \forall ij\in A$) THEN
         infeas ← .true.
     ELSE
         CALL CALC_$\theta$                  !Calculate optimal  value
         $\pi' \leftarrow \pi + \theta\Delta\pi$                !Update dual prices
         $\sigma' \leftarrow \sigma + \theta\Delta\sigma$
     END IF
 END DO

 RETURN
 END.
```

Figure 2.2: Primal-Dual Algorithm

Result #4 shows that θ_i, as defined in equation [2.33], is determined by an **inadmissible** path or an **inadmissible** slack variable which becomes admissible with the newly adjusted dual prices. Let the critical element be p^* if it is a path; otherwise, let it be ij^* if it is a slack variable. Then, the reduced cost of path p^* in the i+1st RP problem is:

$$c'_{p*}{}^k = 0 - (-\Sigma_{ij\in A}\, \Delta\pi_{ij}\delta_{ij}{}^{p*,k} + \Delta\sigma^k) = -\gamma_{p*}{}^k < 0. \quad [2.51]$$

Similarly, the reduced cost of ij^* in the i+1st RP problem is:

$$c'_{ij*} = 0 - (-\Delta\pi_{ij}) < 0. \qquad [2.52]$$

Since the reduced cost of any newly admissible element is always strictly negative (see equations [2.51] and [2.52]), a minimum of one pivot will occur in the i+1st RP problem. Thus, using perturbation if necessary to bar degeneracy, the optimal objective function value for the i+1st RP problem will be strictly less than that for the ith RP problem. ■

The conclusions of Papadimitriou and Steiglitz [42] and Shapiro [47] are adapted here for the CMCF problem as follows:

__Result #7:__ The primal-dual algorithm, described in Figure 2.2, optimally solves the CMCF problem in a finite number of steps.

__Proof:__ If all of the paths in problem P, plus all of the artificial variables are included as variables in the RP problem, then the solution of each RP problem at each iteration of the primal-dual method produces a basic feasible solution to the CMCF problem, P. Result #6 showed that if the ith RP problem's optimal objective function value is strictly greater than zero, then an i+1st RP problem can be solved and its optimal objective function value will be strictly less. This result follows from the Fundamental Property of the Simplex Method [13] wherein it is stated that the simplex method, using perturbation if necessary, solves any given linear program in a finite number of iterations.

Hence, by employing a rule to avoid degeneracy, the iterations of the primal-dual algorithm are guaranteed to produce non-repeating basic feasible solutions to the CMCF problem and thus, the primal-dual algorithm optimally solves the CMCF problem in a finite number of steps. ∎

2.5 Summary

The primal-dual approach is particularly useful when, due to problem size, the solution to P takes an inordinate amount of time to achieve. The idea is to **indirectly** solve a given problem P by **iteratively** solving a **restated** problem, RP; where RP is smaller and simpler than P. Once the solution to the RP problem satisfies a certain condition, optimality in P is achieved. It is guaranteed that the primal-dual method optimally solves problem P in a finite number of steps.

3. Algorithmic Design

This chapter presents the general structure of the new capacitated multi-commodity network flow algorithm.

Section 3.1 presents the motivation for the development of a network-based algorithm to solve the CMCF problem. Section 3.2 explains why the primal-dual approach in particular, was adopted as the basis of the new algorithm. Then Sections 3.3 and 3.4 cast the new algorithm into the framework of the primal-dual method and introduce the algorithmic design features and criterion. Finally, Section 3.5 summarizes the general concepts underlying the algorithmic development.

3.1 Algorithmic Motivation

The development of a new capacitated multi-commodity network flow algorithm is primarily motivated by the need to solve large-scale multi-commodity network flow problems of the type encountered in practice. Numerous applications of this problem type can be found in the transportation and logistics industry, particularly in the areas of vehicle routing and scheduling with freight flow assignment (discussed in Chapter 1). The major weakness of conventional solution methods in solving these real problems is that as problem size increases, formulation size becomes prohibitively large. For example, consider the problem in which a company needs to transport hundreds of shipments from their origins to their respective destinations, using dozens of vehicles. This problem can be formulated as a CMCF problem. Even for problems of this size, however, the CMCF formulation can become difficult to solve because the number of variables grows exponentially with problem size. Specifically, the number of variables for the path formulation of the CMCF problem is equal to the number of commodities multiplied by the number of distinct paths in the network. In a worst case scenario, the number of variables can be as great as $|K|*|N!|$, where $|K|$ is the number of commodities and $|N!|$ is the factorial of the number of nodes in the network. Thus, for the above example, arbitrarily taking $|K| = 100$ and $|N| = 100$, the number of decision variables is on the order of 10^{100}. Thus, the goal is to develop a new algorithm for the CMCF problem which is specifically geared towards solving large network problems of the size typically encountered in industry.

Decomposition and exploitation of special problem structure are two useful strategies in tackling large scale problems. The next section describes how these strategies shape the design of the new algorithm for the CMCF problem.

3.2 Primal-Dual Method vs. Dantzig-Wolfe Method

Chapter 1 presented the Dantzig-Wolfe method and Chapter 2 presented the Primal-Dual method. Both methods solve the original CMCF problem P **indirectly** by iteratively solving a restated problem, RP. RP, in both methods, is smaller in size than the original problem P. (The Dantzig-Wolfe and Primal-Dual RP problems are smaller than P not in the number of rows but instead, in the number of columns. Both the Dantzig-Wolfe and the primal-dual methods generate columns as necessary until optimality is achieved. Problem P, on the other hand, contains all possible columns from the onset.)

In the Dantzig-Wolfe method, RP, although smaller, is formulated exactly as P. In other words, RP is a minimum-cost, capacitated multi-commodity network flow problem. In the primal-dual method, however, although the RP problem is a capacitated multi-commodity network flow problem, its objective is simplified to that of finding a feasible primal solution. The objective function of the primal-dual RP problem does not contain arc costs because they are implicitly considered through the concept of admissibility. This simplified objective of the RP problem is, however, achieved at some expense to the primal-dual method. Table 3.1 shows the major components of the primal-dual and Dantzig-Wolfe methods. The primal-dual method performs a step not included in the Dantzig-Wolfe method. It is this additional step, which determines the amount by which dual prices are adjusted (i.e. Determine θ_i), that enables arc costs to be eliminated from the RP problem.

In comparing and evaluating the primal-dual and Dantzig-Wolfe methods, the question is whether the simpler objective of the RP problem in the primal-dual method as compared to the Dantzig-Wolfe method justifies expending additional time and effort to adjust dual prices. The answer to this question is, of course, dependent on the nature of the problem being solved. In the case discussed here, the types of problems to be solved have corresponding RP problems whose size cannot be easily accommodated by the simplex method.

	<u>Dantzig-Wolfe</u>	<u>Primal-Dual</u>
RP Smaller than P?	YES	YES
RP Simpler than P?	NO	YES
Find Initial Solution?	YES	YES
Solve RP/DRP?	YES	YES
Determine θ_1?	NO	YES
Dual Price Adjustment?	YES	YES
Update RP?	YES	YES

Table 3.1: Primal-Dual vs. Dantzig-Wolfe

As stated previously, the RP problem for both the Dantzig-Wolfe and primal-dual methods is itself a CMCF problem-- one which is reduced in size in comparison to the original CMCF problem, P. Unfortunately, even a reduced-size CMCF problem is laden with difficulties. The multi-commodity aspect introduces numerous interactions not present in the single-commodity case and the coupling constraints prohibit easy decomposition by commodity. The Dantzig-Wolfe method requires that the CMCF subproblem, RP, be optimally solved. The primal-dual method, however, requires only that the CMCF subproblem be solved for a feasible solution (if one exists). It is exactly this simplification that leads to the selection of the primal-dual method, and not the Dantzig-Wolfe method, as the theoretical basis for the new algorithm to solve large-scale CMCF problems. Thus, the answer to the question posed previously is that for large scale problems, the additional work required in determining revised dual prices in the primal-dual method will hopefully be compensated for by the benefit that its RP subproblem is not only smaller than P, but also simpler than the Dantzig-Wolfe RP subproblem.

The new Primal-Dual Network algorithm, termed PDN, is cast in the primal-dual framework and is motivated by the desire to solve large scale minimum cost, capacitated multi-commodity network flow problems. The RP subproblems are solved with an algorithm specifically geared to exploit the network structure of the CMCF problem.

3.3 Algorithmic Framework

The primal-dual method, as presented in Chapter 2, provides the theoretical foundation for the PDN algorithm. Thus, dual feasibility is always maintained by the PDN algorithm. At each iteration, a primal solution is constructed which, although possibly infeasible, always guarantees the satisfaction of certain complementary slackness conditions. The objective is to find, for the given dual solution and resulting set of admissible elements, a primal solution which is as "close" to satisfying the feasibility requirements as possible. The "closeness" of the solution to feasibility is gauged with a feasibility measure which is equal to zero when feasibility is achieved. By construction, optimality is achieved simultaneously with primal feasibility. If at a given iteration, the "best" primal solution is not feasible, the dual solution is modified and the algorithm reiterates.

Figure 3.1 depicts the Primal-Dual Network Algorithm. The structure of the PDN algorithm is designed to parallel that of the standard primal-dual method, shown in Figure 2.1. In fact, there is a one-to-one correspondence between the steps of the PDN algorithm and the steps of the standard primal-dual method. Both algorithms contain the following:

i) Step 0: Generate Initial Dual Solution;

ii) Step 1: Define Admissible Elements;

iii) Step 2: Solve the Restricted Primal and Dual Problem;

iv) Step 3: Test for Optimality; and

v) Step 4: Generate New Solution for D.

In adapting the standard primal-dual method to the specifically tailored PDN algorithm, steps 0, 1, 3 and 4 are essentially unchanged.

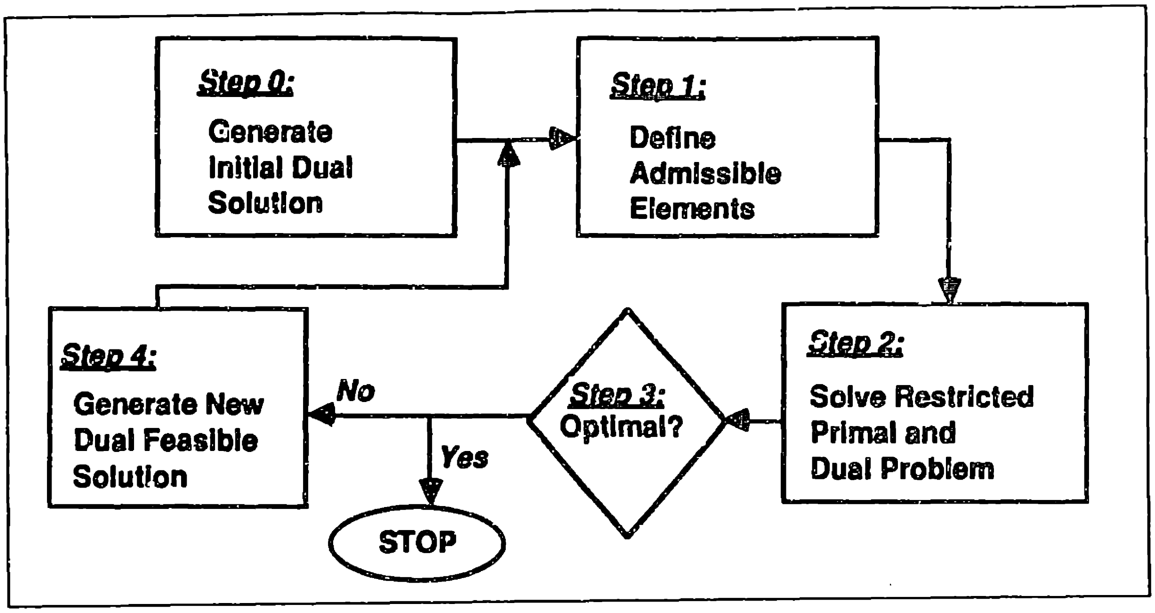
Step 0:
Generate Initial Dual Solution
Step 1:
Define Admissible Elements
Step 2:
Solve Restricted Primal and Dual Problem
Step 3:
Optimal?
No
Yes
Step 4:
Generate New Dual Feasible Solution
STOP

Figure 3.1: The PDN Algorithm

It is only step 2-- the solution of the RP and DRP subproblem, which is significantly altered for the PDN algorithm. As shown in Figure 3.2, instead of solving step 2 with the simplex method as in the standard primal-dual algorithm, the PDN algorithm uses an iterative, network-based solution technique. In the PDN algorithm, the solution of the RP/DRP problem (Step 2) is achieved by repeatedly performing Flow Adjustment Steps using the Flow Adjustment Algorithm. As demonstrated later, these primal-based Flow Adjustment Steps work on the current primal solution and by shifting flow around, move the solution closer to primal feasibility and to optimality. Flow adjustment steps are repeated until no further reduction in infeasibility is possible for the current admissible set and the RP problem is solved to optimality.

If the optimal solution to the RP problem is still infeasible for P, a new solution for D is generated with a dual-based Price Adjustment Step (Step 4). This new dual solution for D, satisfies complementary slackness conditions #2 and #3 for the current flow assignments and increases the dual objective function value.

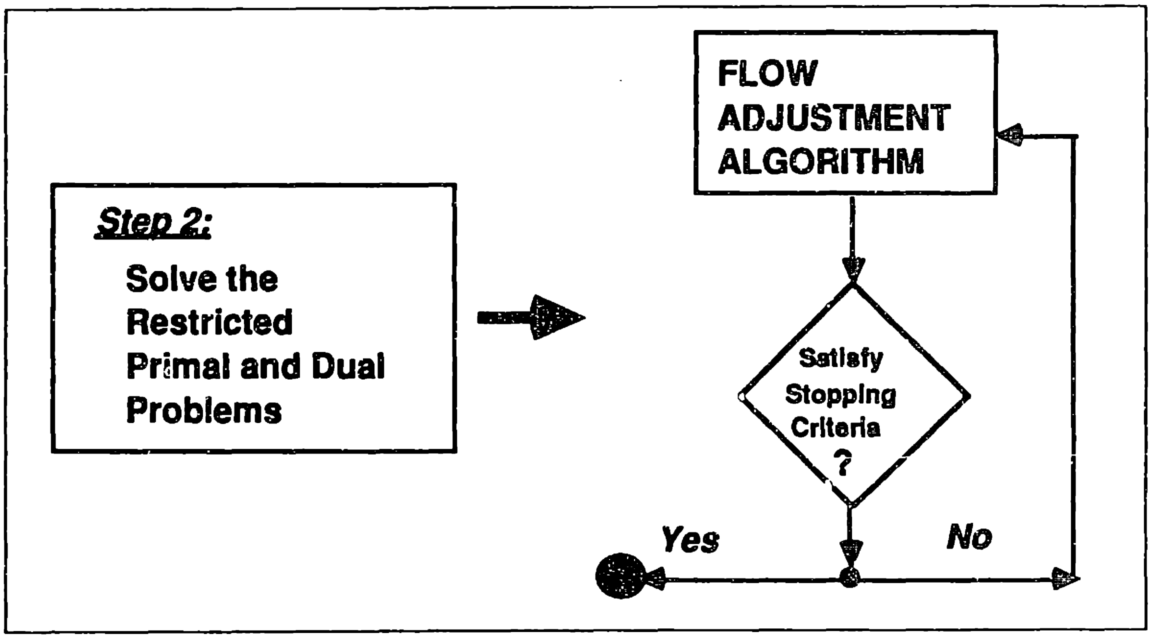
Step 2:
Solve the Restricted Primal and Dual Problems
FLOW ADJUSTMENT ALGORITHM
Satisfy Stopping Criteria ?
Yes
No

Figure 3.2: Solution of the RP/DRP Problem

3.4 Algorithmic Design Features and Criterion

The PDN Algorithm is cast in the primal-dual framework and thus, solves the original CMCF problem P by repeatedly solving a smaller, restated problem RP. RP differs from P not only in size but also in the bundle constraints (equations [2.2]), which are relaxed in RP. To ensure that P will be optimally solved by solving RP, the following design features are included in the PDN algorithm:

#1: Each step of the PDN algorithm must maintain **dual feasibility**;

#2: Each step of the PDN algorithm must maintain **complementary slackness conditions #3** (equations [2.12]), which permits the assignment of flow only to shortest paths;

#3: Each step of the PDN algorithm must satisfy **demand constraints** (primal constraints [2.3]) (thereby satisfying **complementary slackness conditions #2** (equations [2.11]); and

#4: Each step of the PDN algorithm must maintain **flow non-negativity** (primal constraints [2.4]).

Control is exercised by appropriately defining and restricting each step of the algorithm so as to satisfy the above features and the following criterion:

Design Criterion: Measurable advancement towards optimality must be achieved at each iteration of the PDN algorithm.

In summary, the PDN algorithm, modeled after the primal-dual method, alters the primal and dual solutions so that measurable advancement towards optimality in P is achieved. The design features of the PDN algorithm ensure that each altered primal and dual solution do not destroy satisfaction of dual feasibility; complementary slackness conditions #2 and #3; demand requirements; or flow non-negativity.

As in the general primal-dual algorithm, the PDN algorithm terminates when the RP objective function value is reduced to zero. At this point, by Result #2 and the stated features of the PDN algorithm, feasibility in P and D, as well as the complementary slackness conditions are all satisfied and hence, optimality is achieved.

3.5 Summary

The need to solve large-scale CMCF problems of the size encountered in practice motivated the development of a new network-based primal-dual algorithm, called PDN. The primal-dual methodology was adopted as the framework for the PDN algorithm because:

i) it employs decomposition and therefore never directly solves the original (very large) problem P; and

ii) the RP subproblems are both smaller in size and simpler in form than P.

Although reduced in size, the RP subproblems are not small enough to be efficiently solved (or solved at all) using general LP-based methods. Instead, the PDN algorithm uses a specialized network based method to solve the RP subproblems. For this reason, the PDN algorithm is modeled after the primal-dual method, which has a simplified RP subproblem.

Chapters 4 and 5 describe in detail the design of the network-based technique used in the PDN algorithm to solve the RP/DRP subproblems.

4. Solution Method for the RP and DRP Problems

The PDN Algorithm solves the CMCF problem indirectly by iteratively solving restricted primal and dual problems, denoted RP and DRP, respectively (as shown in Figure 3.2). The RP/DRP problems can be solved with the simplex method if problem size permits. However, the simplex method is not amenable to very large scale problems. Thus, the PDN algorithm uses a network-based method, referred to as the Flow Adjustment Algorithm, to solve these subproblems. The next two chapters are devoted to describing this network based algorithm. Chapter 4 first presents the inherent challenges of solving the RP/DRP subproblems for the CMCF problem. Then, Chapter 5 describes the steps of the Flow Adjustment Algorithm in detail.

Section 4.1 restates the formulations of the RP and DRP problems. Next, section 4.2 introduces definitions useful in describing the flow adjustment step performed by the Flow Adjustment Algorithm. Then, sections 4.3 and 4.4 provide insight into the implementation difficulties. Section 4.5 shows that these difficulties are attributable to the multi-commodity aspect of the problem. Finally, Section 4.6 summarizes the general strategy used in the network-based Flow Adjustment Algorithm to solve the RP subproblems.

4.1 The RP and DRP Formulations Restated

As presented in Chapter 2, the adopted formulation for the RP problem is as follows:

$$\text{MIN } z_{RP} = \Sigma_{ij\in A} (w_{ij}^- + w_{ij}^+) \qquad [4.1]$$

subject to:

$$\Sigma_{k\in K} \Sigma_{p\in J^k} x_i^k \delta_{ij}^{i,k} + w_{ij}^+ - w_{ij}^- = d_{ij}, \quad \forall\ ij\in IJ^+(\pi); \qquad [4.2a]$$

$$\Sigma_{k\in K} \Sigma_{p\in J^k} x_i^k \delta_{ij}^{p,k} + s_{ij} - w_{ij}^- = d_{ij}, \quad \forall\ ij\in IJ^+(\pi)^c; \qquad [4.2b]$$

$$\Sigma_{p\in J^k} x_p^k = b^k, \quad \forall\ k\in K; \qquad [4.3]$$

$$x_p^k \geq 0,\ \forall\ p\in J^k,\ \forall\ k\in K;\ x_p^k = 0,\ \forall\ p\in (J^k)^c,\ \forall\ k\in K; \qquad [4.4]$$

$$w_{ij}^-,\ w_{ij}^+ \geq 0,\ \forall\ ij\in A;\ w_{ij}^+ = 0,\ \forall\ ij\in IJ^+(\pi)^c. \qquad [4.5a]$$

$$s_{ij} \geq 0,\ \forall\ ij\in A;\ s_{ij} = 0,\ \forall\ ij\in IJ^+(\pi). \qquad [4.5b]$$

where:

J^k: set of least-cost (shortest) paths in set J for commodity $k\in K$;

$(J^k)^c$: complement of set J^k;

$IJ^+(\pi)$: set of arcs $ij\in A$ with $\pi_{ij} > 0$;

$IJ^+(\pi)^c$: complement of set $IJ^+(\pi)$;

w_{ij}^+, w_{ij}^-: artificial variables measuring the infeasibility of the current flow assignment on each arc $ij\in A$.

The dual of the RP problem, DRP, is as follows:

$$\text{MAX } z_{DRP} = \Sigma_{ij\in A} -\Delta\pi_{ij} d_{ij} + \Sigma_{k\in K} \Delta\sigma^k b^k \qquad [4.6]$$

subject to:

$$-\Sigma_{ij\in A} \Delta\pi_{ij} \delta_{ij}^{p,k} + \Delta\sigma^k \leq 0, \quad \forall\ p\in J^k,\ \forall\ k\in K; \qquad [4.7]$$

$$-1 \leq \Delta\pi_{ij} \leq 1, \qquad \forall\ ij \in IJ^{+}(\pi); \qquad [4.8]$$

$$0 \leq \Delta\pi_{ij} \leq 1, \qquad \forall\ ij \in IJ^{+}(\pi)^{c}; \qquad [4.9]$$

$$\Delta\sigma^{k} \text{ u.i.s.}, \qquad \forall\ k \in K. \qquad [4.10]$$

4.2 Definitions

Definition 4.1: An **over-capacitated** arc, denoted OCarc, is an arc with excess flow, or equivalently, an arc with flow exceeding capacity. An over-capacitated arc is categorized as a "problem" arc because its flow violates the (primal) arc capacity constraint (equations [2.2]). The amount of over-capacitation on arc $ij \in A$, oc_{ij}, is determined as follows:

$$oc_{ij} = \text{MAX}(0,\ X_{ij} - d_{ij}), \qquad \forall\ ij \in A. \qquad [4.11]$$

Definition 4.2: An **under-capacitated** arc, denoted UCarc, is an arc with a non-zero dual arc price and flow strictly less than capacity. Under-capacitated arcs, like over-capacitated arcs, are categorized as problem arcs. UCarcs do not satisfy arc capacity constraints (equations [2.2]) at equality because, by complementary slackness conditions #4 (equation [2.13]), slack on arcs with non-zero dual prices is restricted to zero.

The amount of under-capacitation on an arc $ij \in A$ is expressed as follows:

$$uc_{ij} = \begin{cases} \text{MAX}(0,\ d_{ij} - X_{ij}), & \forall\ ij \in IJ^{+}(\pi); \\ 0, & \forall\ ij \in IJ^{+}(\pi)^{c}. \end{cases} \qquad [4.12]$$

Theorem 4.1: $oc_{ij} = w_{ij}^-$ and $uc_{ij} = w_{ij}^+$.

Proof: From the form of the objective function (equation [4.1]) and constraints [4.2] in the RP problem, the following relationships are observed:

$$w_{ij}^+ > 0 \Rightarrow w_{ij}^- = 0, \quad \forall\ ij \in A;$$
$$w_{ij}^- > 0 \Rightarrow w_{ij}^+ > 0, \quad \forall\ ij \in A; \qquad [4.13]$$

Using these relationships together with constraints [4.2], w_{ij}^+ and w_{ij}^- can be defined as follows:

$$w_{ij}^- = \mathrm{MAX}(0,\ \Sigma_{k\in K}\ \Sigma_{p\in J^k}\ x_p^k \delta_{ij}^{p,k} - d_{ij});\ \forall\ ij \in A. \qquad [4.14]$$
$$w_{ij}^+ = \{\mathrm{MAX}\ (0,\ d_{ij} - \Sigma_{k\in K}\ \Sigma_{p\in J^k}\ x_p^k \delta_{ij}^{p,k}),\ \forall\ ij \in IJ^+(\pi);$$
$$= 0,\quad \forall\ ij \in IJ^+(\pi)^c\ \}. \qquad [4.15]$$

By inspection, equation [4.14] is equivalent to equation [4.11] and equation [4.15] is equivalent to equation [4.12]. ■

Thus, the artificial variables w_{ij}^+ and w_{ij}^-, $\forall\ ij\in A$ can be interpreted as measures of primal infeasibility. Given this interpretation, the objective of the RP problem is to find the flow assignment which minimizes the number of violations to feasibility in P.

The PDN algorithm requires the satisfaction of all the primal feasibility constraints (equations [2.2] through [2.4]) except the arc capacity constraints (equations [2.2]). Thus, the measure of primal infeasibility (denoted I and defined below), is equal to the total amount by which the primal feasibility constraints [2.2] are violated.

Constraints [2.2] are violated if arc flow exceeds capacity or if arc flow is less than capacity and slack on the arc is restricted to zero. These two situations define OCarcs and UCarcs, respectively. Thus, the infeasibility measure is defined as follows:

$$I = \Sigma_{ij \in A} (oc_{ij} + uc_{ij}). \qquad [4.16]$$

From Theorem 4.1, the infeasibility measure can be equivalently expressed as:

$$I = \Sigma_{ij \in A} (w_{ij}^{-} + w_{ij}^{+}). \qquad [4.17]$$

By equivalence of equations [4.1] and [4.17], it follows that:

$$I = z_{RP} \qquad [4.18]$$

Result #2 and equation [4.18] lead to the following corollary:

<u>Corollary 4.1:</u> The CMCF problem P is optimally solved if the infeasibility measure is equal to zero. ∎

Thus, the objective of the RP problem is to find a flow assignment which minimizes the total primal infeasibility, given that the only possible source of infeasibility is violation of the capacity constraints (represented by equations [4.2] in the RP problem). In the PDN algorithm, this objective is obtained by repeatedly performing flow adjustment steps (Step 2 of the PDN Algorithm). The primal-based flow adjustment step improves the primal (infeasible) solution by moving it closer to feasibility while satisfying the algorithmic design features. These criteria are satisfied by defining the <u>FLOW ADJUSTMENT STEP</u> as follows:

<u>Definition 4.3:</u> The flow adjustment step shifts existing commodity flows among shortest paths so as to achieve a net decrease in the total amount of primal infeasibility.

Any flow shift satisfying the requirements of Definition 4.3 is successful in satisfying all of the design features of the PDN algorithm. Specifically:

i) **Dual feasibility** (equations [2.6] through [2.8]) is unaffected because flow adjustments modify only the primal solution and not the dual solution;

ii) **Demand constraints** (equations [2.3]) are guaranteed to be satisfied because flow adjustments shift flow of a particular commodity between **paths**;

iii) **Complementary slackness conditions #3** (equations [2.12]) are satisfied because flow adjustments shift flow between **shortest** paths; and

iv) **Primal non-negativity constraints** (equations [2.4]) are satisfied because flow adjustments shift only **existing** flow.

Hence, the flow adjustment step is designed to shift flow between shortest paths without violating any of the algorithmic design features. Furthermore, the flow adjustment step is designed to shift flows only if a net decrease in the total amount of primal infeasibility, I, is achieved. By Corollary 4.1, optimality is achieved when

$I=0$. Thus, each flow adjustment step strictly reducing I represents advancement (measured in terms of the amount of reduction in I) towards optimality. Hence, if a potential flow shift meets the criteria of the flow adjustment step, it is executed and the design criterion of the PDN algorithm is satisfied.

<u>Theorem 4.2:</u> If optimality in P is achieved for the current flow assignments, then $I = 0$.

<u>**Proof:**</u> If optimality in P is achieved:

i) from equations [2.2] and [4.2]:

$$w_{ij}^{-} = 0, \quad \forall\ ij \in A; \qquad [4.19a]$$

ii) from equations [4.5b] and [2.2]:

$$s_{ij} = 0 \text{ and } w_{ij}^{+} = 0, \quad \forall\ ij \in IJ^{+}(\pi) \qquad [4.19b]$$

and thus, complementary slackness conditions #4 (equations [2.13]) are satisfied; and

iii) by equations [4.5a]:

$$w_{ij}^{+} = 0, \quad \forall\ ij \in IJ^{+}(\pi)^{c}. \qquad [4.19c]$$

Using equation [4.17] together with equations [4.19]:

$$I = 0. \qquad [4.20]$$

∎

From Result #2 and Theorems 4.1 and 4.2, the following corollary results:

<u>Corollary 4.2:</u> The CMCF problem P is optimally solved if and only if no OCarcs and no UCarcs exist for the current flow assignment. ▨

To summarize, each flow adjustment step searches for a flow shift which satisfies the algorithmic design features and, at the same time, decreases the value of the infeasibility measure.

Analogies can be drawn between the execution of a flow adjustment step and a pivot in the simplex method. When the simplex method is used to solve the RP problem, each non-degenerate pivot reduces the value of z_{RP}. Similarly, in the PDN algorithm, each flow adjustment step is designed to decrease the value of I ($= z_{RP}$). In the simplex method, pivots are performed for a given RP problem as long as they result in a decrease in z_{RP}. Similarly, in the PDN algorithm, flow adjustments are repeated as long as they achieve a reduction in I.

In the PDN algorithm, the RP problem is solved by first selecting a problem arc, either an OCarc or an UCarc. Then, based on the rationale that optimality is achieved with the elimination of OCarcs and UCarcs (Corollary 4.2), the PDN algorithm searches for a flow adjustment (among shortest paths) which removes flow from the selected arc if it is an OCarc or adds flow to the selected arc if it is an UCarc; and decreases the value of I.

<u>4.3 Search Process</u>

First, to simplify the scope of the search for a flow adjustment satisfying definition 4.3, observe that any flow shift removes flow of a single commodity off one shortest "from-path" and places that flow onto another shortest "to-path". Thus, the flow adjustment search can be performed in a restricted network containing only shortest paths. Furthermore, the search can be simplified by observing that the from-path and the to-path, although distinct, have at least two nodes in common-- namely the origin and destination nodes of the commodity shifted. Thus, at least one cycle, denoted Γ, is formed by the two paths. The purpose of the flow shift between the two paths is to reduce the infeasibility measure I. Hence, the flow shift between the from and to-paths can be effectively represented as a flow shift around the cycle Γ (Figure 4.1), where flow is added to the arcs in Γ on the to-path and flow is removed from the arcs in Γ on the from-path. The search for this cyclic flow shift is simplified when a residual network is used. Thus, each commodity-specific flow shift is performed on a residual restricted network for that commodity, denoted RN^k.

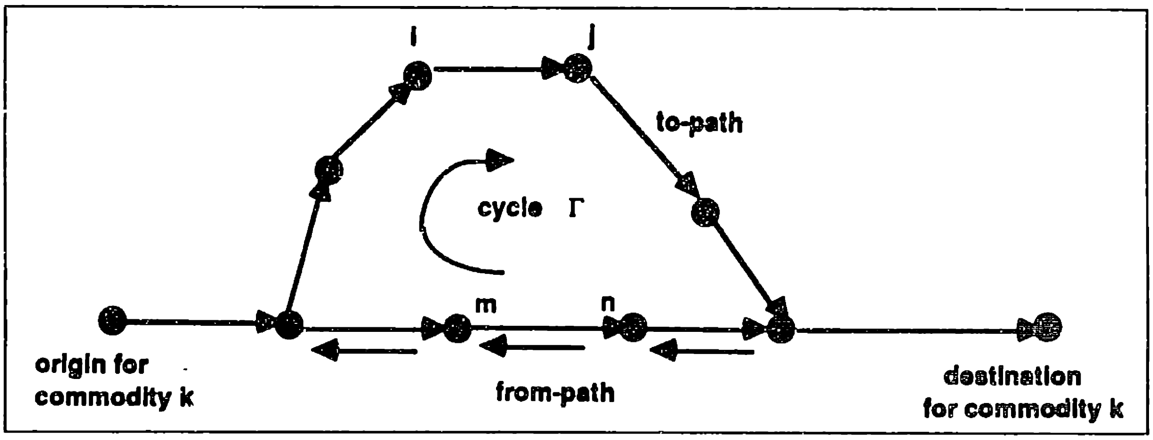

Figure 4.1: Cycle Flow Shift

Definition 4.4: A **residual restricted network** for commodity k, RN^k, is a network containing only shortest paths for commodity k. In RN^k, each arc $ij \in A$ on a shortest path, is replaced with two arcs: a forward arc from node i to node j and a reverse arc from node j to node i. The costs, υ_{ij}, on the arcs in RN are defined as follows:

$$\begin{aligned}
\upsilon_{ij} &= 1, \text{ if } ij \in F; \ X_{ij} \geq d_{ij}; \\
&= -1, \text{ if } ij \in F; \ X_{ij} < d_{ij}, \ \pi_{ij} > 0; \\
&= 1, \text{ if } ij \in F^c, \ X_{ji} \leq d_{ij}, \ \pi_{ji} > 0; \\
&= -1, \text{ if } ij \in F^c, \ X_{ji} > d_{ji}; \\
&= 0, \text{ otherwise.}
\end{aligned} \qquad [4.21]$$

where:

F: set of forward arcs in RN;

F^c: complement of set F-- the set of reverse arcs in RN.

Let RN denote the "aggregate" residual restricted network containing every arc in each commodity specific network. Rather than replicating information throughout each RN^k network, quantities regarding arc flow and dual arc price are stored on the arcs in RN. (Although flow shifts of commodity k are technically performed over the commodity specific network RN^k, for ease of description, the flow shifts will simply be described as occurring in the aggregate network RN).

The following standard notation is introduced to simplify the explanation process:

ij^a: "conceptual" aggregate arc achieved by combining the forward arc from node i to node j and the reverse are from node j to node i.

X_{ij^a}, $x_{ij^a}^k$: total flow and commodity-specific flow on arc $ij^a \in RN$ where:

$$x_{ij^a}^k = x_{ij}^k - x_{ji}^k; \qquad \forall\ ij,\ ji,\ ij^a \in RN; \qquad [4.22a]$$

$$X_{ij^a} = X_{ij} - X_{ji}, \qquad \forall\ ij,\ ji,\ ij^a \in RN. \qquad [4.22b]$$

The arc capacities, denoted $d_{..}$, are as follows:

$$\begin{aligned} d_{ij^a} &= d_{ij}, && \forall\ ij \in A,\ ij^a \in RN; \\ d_{ij} &= \infty, && \forall\ ij \in RN^k \cap F,\ \forall\ k \in K; \\ d_{ij}^k &= x_{ji^a}^k, && \forall\ ij \in RN^k \cap F^c,\ \forall\ k \in K. \end{aligned} \qquad [4.23]$$

where:

d_{ij}: capacity of arc $ij \in A$;

d_{ij}^k: capacity of the reverse arc $ij \in RN^k$, specified for each commodity $k \in K$.

Defining arc capacities in this manner ensures that flow non-negativity is always maintained. The reverse arc capacities, which are specified by commodity, restrict the amount of flow of a commodity removed from an arc to the amount of existing flow of that commodity on the arc. Thus, if commodity k is not assigned to forward arc $ij \in RN$, then the capacity of the corresponding reverse arc in RN^k is equal to zero and the reverse arc, in essence, does not exist for commodity k.

As with the flow and capacity of the original arc $ij \in A$, information concerning the dual arc price, π_{ij}, is contained on the aggregate arc $ij^a \in RN$.

As shown in Figure 4.1, a flow shift of commodity k around a directed cycle in RN, denoted Γ, can be interpreted as a flow shift removing flow from one shortest path and placing it onto another shortest path. The arcs on the paths from which flow is removed are represented by reverse arcs in the directed cycle. Similarly, the arcs on the path to which flow is added are represented by forward arcs in the directed cycle. Thus, a flow shift around Γ of commodity $k \in K$ adds flow to forward arc $ij \in RN$ if the arc from node i to node j is contained in Γ and removes flow from reverse arc $nm \in RN$ if the arc from node j to node i is contained in Γ.

Define the cost of cycle Γ, Ψ_Γ, as the sum of the costs on the arcs contained in the cycle, i.e.:

$$\Psi_\Gamma = \Sigma_{ij \in \Gamma} \, \nu_{ij}, \quad \text{for cycle } \Gamma. \qquad [4.24]$$

Then, let Δ^* be the maximum size flow shift around cycle Γ which satisfies the following conditions:

i) the total flow on an over-capacitated aggregate arc is reduced at most to the capacity of the arc; and

ii) the total flow on an aggregate arc with flow strictly less than capacity is increased at most to the capacity of the arc.

__Theorem 4.3:__ In the residual network with arc costs as defined in equations [4.21], the cycle cost (equation [4.24]) represents the change in the infeasibility measure per unit of flow shifted around the cycle for a flow shift of size Δ^*, i.e.:

$$\Delta I = \Delta^*(\Psi_\Gamma). \qquad [4.25]$$

__Proof:__ Given the restriction on the size of Δ^*, a flow shift of Δ^* units around cycle Γ increases the value of the infeasibility measure only if:

i) flow is increased on arc $ij^a \in RN$ with $X_{ij}a \geq d_{ij}a$; or

ii) flow is decreased on arc $ij^a \in RN$ with $X_{ij}a \leq d_{ij}a$ and $\pi_{ij}a > 0$.

Similarly, the value of infeasibility is decreased by such a flow shift if:

i) flow is decreased on arc $ij^a \in RN$ with $X_{ij}a > d_{ij}a$; or

ii) flow is increased on arc $ij^a \in RN$ with $X_{ij}a < d_{ij}a$ and $\pi_{ij}a > 0$.

Thus, the change in the infeasibility measure brought about by a flow shift of Δ^* units around cycle Γ can be expressed using the arc costs described in equations [4.21] as follows:

$$\Delta I = \Sigma_{ij \in \Gamma} (\nu_{ij})\Delta^* \qquad [4.26]$$

From equations [4.25] and [4.26], it follows that:

$$\Delta I = \Delta^* \Psi_\Gamma. \qquad [4.27]$$

■

Notice that ΔI can also be defined as:

$$\Delta I = I^{i+1} - I^{i} \qquad [4.28]$$

where:

I^i: value of the infeasibility measure prior to the i+1st flow shift.

Similarly, the change in the value of the RP objective function resulting from a flow shift of Δ^* around cycle Γ, denoted Δz_{RP}, can be expressed as:

$$\Delta z_{RP} = z_{RP}^{i+1} - z_{RP}^{i} \qquad [4.29]$$

where:

z_{RP}^i: objective function value of the RP problem prior to the i+1st flow shift.

The following corollary results from Theorem 4.3 and equation [4.18]:

<u>Corollary 4.3:</u> $\Delta z_{RP} = \Delta^*(\Psi_\Gamma)$ [4.30]

■

The idea then, is to specify each arc's cost so as to capture the change in the infeasibility measure (or equivalently, the change in the objective function value of RP) realized by adding or removing flow from the arc. Then, by Theorem 4.3, the flow adjustment goal to find a flow shift reducing the value of the infeasibility measure can be achieved by sending Δ^* units of flow around a directed, negative cost cycle in RN. In other words, the infeasibility measure can be reduced if there is a negative cost cycle in RN. The difficulty is that the opposite of this result does not hold. Indeed, even if there are no negative cost cycles in the residual network, the infeasibility measure may be reduced through a particular type of flow shift in RN. This "special" flow shift is a composite flow shift containing several individual, commodity-specific flow shifts. Each individual flow shift moves flow of some commodity around a non-negative cost cycle in the corresponding RN^k network and the cycles combine together in RN to form a composite cycle. The flow shifts around the individual cycles in the composite cycle interact to form a composite flow shift which reduces the infeasibility measure.

For example, Figure 4.2 shows a composite flow shift containing cycle #1, denoted c1, and cycle #2, denoted c2. Cycle c1 has cost $\Psi_{c1} = 0$ and cycle c2 cost $\Psi_{c2} = 1$. Cycle shift #1, taken alone, shifts Δ^* units of some commodity around cycle c1, thereby reducing the flow on OCarc A at the expense of over-capacitating arc B. Cycle shift #2, taken alone, shifts Δ^* units of some commodity around cycle c2 and therefore, under-capacitates arc B. However, when cycle shifts #1 and #2 are combined, the over-capacitating effects of cycle shift #1 and the under-capacitating effects of cycle shift #2 on arc B are nullified. The net result is that the composite flow shift removes flow from OCarc A and improves the current solution by achieving a net decrease in the infeasibility measure, i.e. $\Delta I < 0$.

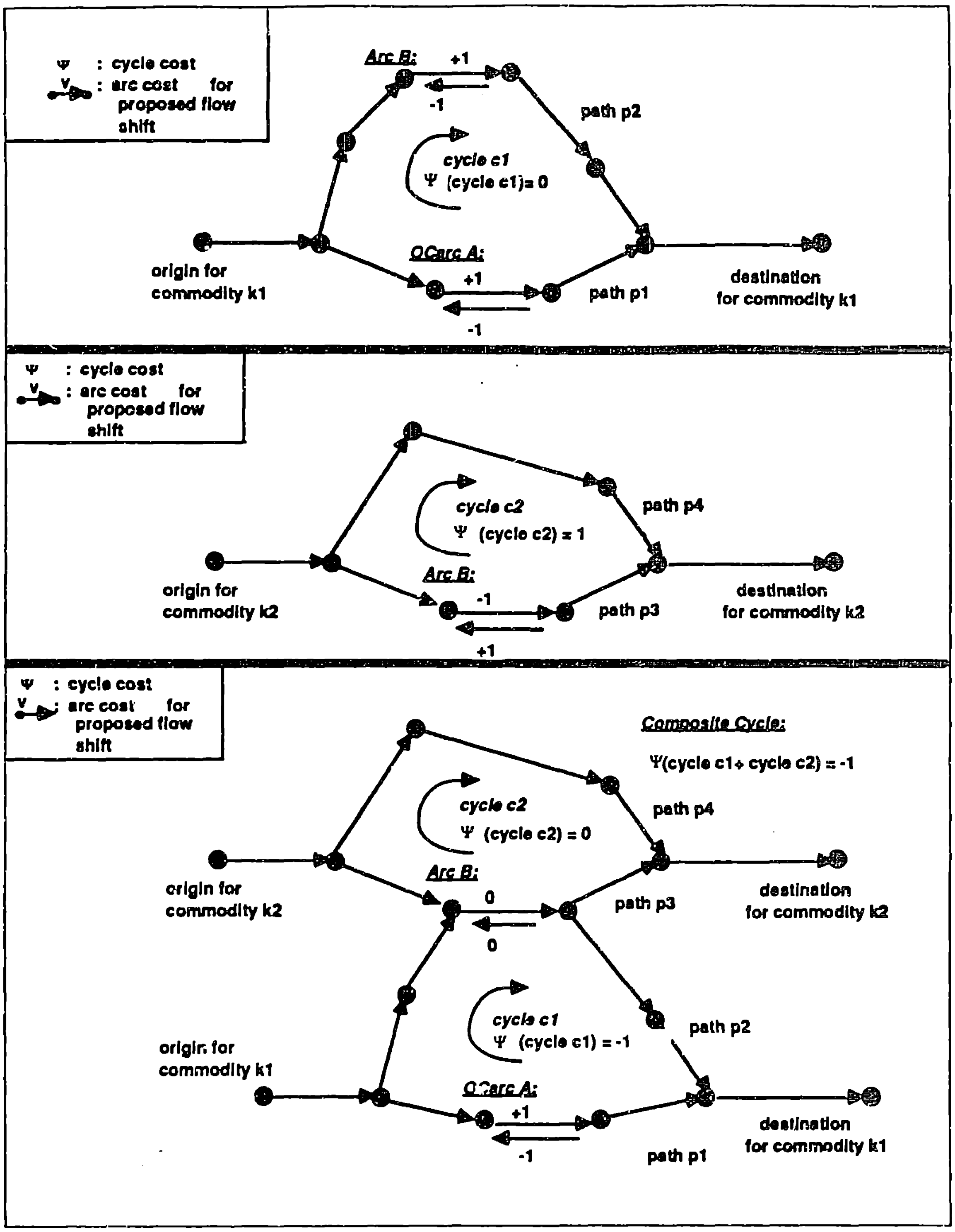

Figure 4.2: Composite Flow Shift

A decrease in the infeasibility measure results even though all the individual cycles comprising the composite cycle have non-negative cost. This occurs because, unlike flows, cycle costs are not additive. The individual cycle costs do not sum to the net cycle cost for the composite cycle. This is best explained through an example.

Again refer to Figure 4.2 and focus on arc B whose total flow, before any flow shift, is exactly equal to arc capacity. In the composite flow shift, the net change in flow on arc B is zero and thus, the resulting contribution to the infeasibility measure is also equal to zero. From Theorem 4.3, it follows that the net contribution for arc B obtained by adding its cost in cycle c1 together with its cost in cycle c2 should likewise be zero. However, in cycle c1, arc B has cost = 1, since $X_Ba \geq d_Ba$. In cycle c2, the reverse arc for B has cost = +1 since $X_Ba \leq d_Ba$, $\pi_Ba > 0$. Thus, arc B contributes a total cost of 2, and not 0, to the value of the cost of the cycles comprising the composite cycle. The discrepancy results because arc costs are different (as defined in equation [4.21]) depending on the total amount of flow on the arc. For example, the cost for the reverse arc for B is equal to +1 if total flow on B is exactly at capacity but is equal to -1 if arc B is over-capacitated.

Thus, cycle costs exactly measure the contribution to the change in the infeasibility measure for an individual cycle shift but do not exactly measure the contribution to ΔI of a combination of flow shifts. This presents a rather serious problem. It shows that the arc costs defined in equations [4.21] are valid only for flow shifts containing a single cycle.

4.4 Arc Costs Revisited

In the multiple-commodity arena, it may be necessary to combine several cyclic flow shifts in order to reduce the infeasibility measure. In the search for and construction of this composite flow shift, the total flow on some arcs is repeatedly altered and thus, costs must be continuously updated to accurately measure the change in infeasibility achievable for the flow shift. The arc costs defined in equations [4.31] below are determined by rewriting equations [4.21] to account for:

i) the explicit bounding of the flow shift size; and

ii) the possibility that the flow shift may consist of several commodity-specific cycle flow shifts.

OCARCS:

ν_{ij} = 1, if $ij \in F$; $X_{ij}a > d_{ij}a$;

= -1, if $ij \in F^c$, $X_{ji}a > d_{ji}a$; $\Delta^{NET}{}_{ji}a \leq X_{ji}a - d_{ji}a$;

UCARCS:

= -1, if $ij \in F$; $X_{ij}a < d_{ij}a$, $\pi_{ij}a > 0$; $\Delta^{NET}{}_{ij}a \leq d_{ij}a - X_{ij}a$;

= 1, if $ij \in F^c$, $X_{ji}a < d_{ji}a$, $\pi_{ji}k > 0$;

CAPACITATED ARCS:

$$\begin{aligned}
\upsilon_{ij} &= -1, \text{ if } ij \in F^c,\ X_{ji}a = d_{ji}a,\ \Delta^{NET}{}_{ji}a > 0; \\
&= -1, \text{ if } ij \in F,\ X_{ij}a = d_{ij}a,\ \pi_{ij}a > 0,\ \Delta^{NET}{}_{ij}a < 0; \\
&= 1, \text{ if } ij \in F^c,\ X_{ji}a = d_{ji}a,\ \pi_{ji}a > 0,\ \Delta^{NET}{}_{ji}a < 0; \\
&= 0, \text{ if } ij \in F,\ X_{ij}a = d_{ij}a,\ \pi_{ij}a = 0,\ \Delta^{NET}{}_{ij}a < 0; \\
&= 0, \text{ if } ij \in F^c,\ X_{ji}a = d_{ji}a,\ \pi_{ji}a = 0,\ \Delta^{NET}{}_{ji}a < 0; \\
&= 0, \text{ if } ij \in F,\ X_{ij}a = d_{ij}a,\ \Delta^{NET}{}_{ij}a = 0; \\
&= 0, \text{ if } ij \in F^c,\ X_{ji}a = d_{ji}a,\ \Delta^{NET}{}_{ji}a = 0. \qquad [4.31]
\end{aligned}$$

where:

$\Delta^{NET}{}_{ij}$: the change in total flow on arc $ij \in RN$ due to the composite flow shift;

$\Delta^{NET}{}_{ij}a$: the change in total flow on aggregate arc $ij^a \in RN$ due to the composite flow shift, where:

$$\Delta^{NET}{}_{ij}a = \Delta^{NET}{}_{ij} - \Delta^{NET}{}_{ji},\ \forall\ ij^a \in RN. \qquad [4.32]$$

Equations [4.31] show that arc costs are a function both of the total flow on an arc and the change in the total flow on an arc brought about by the flow shift. As the composite flow shift is constructed, the change in total flow on an arc is altered. Thus, there is not a "static" set of arc costs which can be used from start to finish in the search for a composite flow shift with negative ΔI value. However, the determination of arc costs is somewhat simplified by appropriately bounding the size of the flow shift. With this bounding, some of the arc costs can be statically set.

Take for example an OCarc $ij \in RN \cap F$. As long as the composite flow shift in RN does not reduce the total flow on arc ij below capacity, the infeasibility measure is increased by one unit for each unit of flow shifted onto OCarc ij, and is decreased by one unit for each unit of flow shifted off OCarc ij. This corresponds to setting the cost of forward OCarc $ij \in RN$, υ_{ij}, to +1 and setting the cost of the corresponding reverse arc ji, υ_{ji}, to -1.

A similar analysis applies to the case of an UCarc, denoted $ij \in RN \cap F$. As long as the flow shift in RN does not over-capacitate arc ij, there is a one-to-one correspondence between the amount of flow shifted and the change in ΔI. For each unit of flow shifted onto UCarc ij, the infeasibility measure is decreased by one unit and, for each unit of flow shifted off UCarc ij, the infeasibility measure is increased by one unit. These changes are represented in RN by setting the cost of UCarc ij in RN, υ_{ij}, to -1 and setting the cost of the corresponding reverse arc, υ_{ji}, to +1.

In the case where the aggregate arc $ij^a \in RN$ has total flow strictly less than capacity and dual price equal to zero, arc costs can also be fixed. This time, as long as the amount of flow shifted is bounded so that forward arc ij does not become over-capacitated, the infeasibility measure is unchanged for each unit of flow shifted on or off arc ij. The costs on the forward and corresponding reverse arcs in RN, ij and ji respectively, are set to zero.

Thus, for arcs with flow strictly greater than or strictly less than capacity, arc costs can be fixed in RN provided that each flow shift increases or decreases (whichever may be the case) the total amount of flow on each arc at most to the capacity of the arc.

Now analyze the case of arc $ij^a \in RN$ with total assigned flow exactly equal to capacity. If a flow shift in RN increases the flow on forward arc ij, the costs on arcs ij and $ji \in RN$ should be set as in the OCarc case. However, if a flow shift decreases the flow on arc ij, then the costs on arcs ij and $ji \in RN$ should be set in accordance with the cost specifications for UCarcs.

Thus, for arcs with flow exactly at capacity, arc costs must be updated as cycle shifts are added to or removed from the composite flow shift. All other arc costs, however, can be fixed at the start of the search process.

To summarize, given a particular composite flow shift and appropriate bounds on the size of the flow shift, arc costs (defined by equations [4.31]) precisely measure the per unit change in infeasibility resulting from each unit of flow shifted.

<u>Theorem 4.4:</u> For any <u>composite</u> flow shift in RN (where arc costs are determined using equations [4.31]), the sum (for each cycle in the composite flow shift) of the product of the flow quantity shifted and the cycle cost is equal to the change in the infeasibility measure brought about by the flow shift. This is mathematically stated as follows:

$$\Delta I = \Delta^*(\Sigma_{\Gamma \in \Omega} \Psi_\Gamma) \qquad [4.33]$$

where:

Δ^*: amount of flow shifted for each commodity-specific flow shift;

Ω: set of individual cycles Γ contained in the composite flow shift.

Proof: By definition, the cost of cycle Γ is expressed as:

$$\Psi_\Gamma = \Sigma_{ij\in\Gamma}\, \nu_{ij}. \qquad [4.34]$$

The arc costs defined in equations [4.31] represent the change in the infeasibility measure per unit flow shifted onto arc $ij\in RN$ given an appropriately bounded flôw shift which does not increase the flow on an UCarc above capacity and does not decrease the flow on an OCarc below capacity. Let the total change in flow on arc $ij\in RN$ resulting from the composite flow shift be expressed as:

$$\Delta^{NET}{}_{ij} = \Sigma_{\Gamma\in\Omega}(\Delta^*\delta_{ij}{}^{\Gamma}), \qquad [4.35]$$

where:

$\delta_{ij}{}^{\Gamma}$: $= 1$ if arc $ij\in RN$ is contained in cycle Γ; and $= 0$ otherwise.

Then the change in the infeasibility measure associated with the composite flow shift can be expressed as:

$$\Delta I = \Sigma_{ij\in RN}\, \Delta^{NET}{}_{ij}(\nu_{ij}). \qquad [4.36]$$

Rewriting equation [4.36] using equation [4.35]:

$$\Delta I = \Sigma_{\Gamma \in \Omega} \Sigma_{ij \in RN} (\Delta^{*} \delta_{ij}{}^{\Gamma} \upsilon_{ij}). \qquad [4.37]$$

Substituting equation [4.34] into equation [4.37]:

$$\Delta I = \Sigma_{\Gamma \in \Omega} (\Delta^{*} \Psi_{\Gamma}). \qquad [4.38]$$

Equation [4.33] is obtained by rewriting equation [4.38]. ■

The next sections describe the difficulties resulting when arc costs depend upon arc flows, as in equations [4.31].

4.4.1 Negative Cost Cycles in RN

Theorem 4.3 shows that a reduction in infeasibility can be achieved with a flow shift around a negative cost cycle. The existence of a negative cost cycle does not guarantee however, that an infeasibility-reducing flow shift is possible. In fact, a flow shift around a negative cost cycle may increase the infeasibility measure because arc costs are a function of arc flow and can therefore change as flow is shifted around the cycle. Specifically, the costs on arcs with flow exactly at capacity must be continually updated as flow levels change. The difficulty is that the simultaneous updating of all arc costs leads to cycling in the search for a infeasibility reducing composite cycle shift.

Consider, for example, the negative cost cycle c1 depicted in Figure 4.3a. A flow shift of Δ^{*} units around cycle c1 results in the over-capacitation of arcs ij, kl, and mn. To accurately reflect their contribution to the change in the infeasibility measure, the costs of

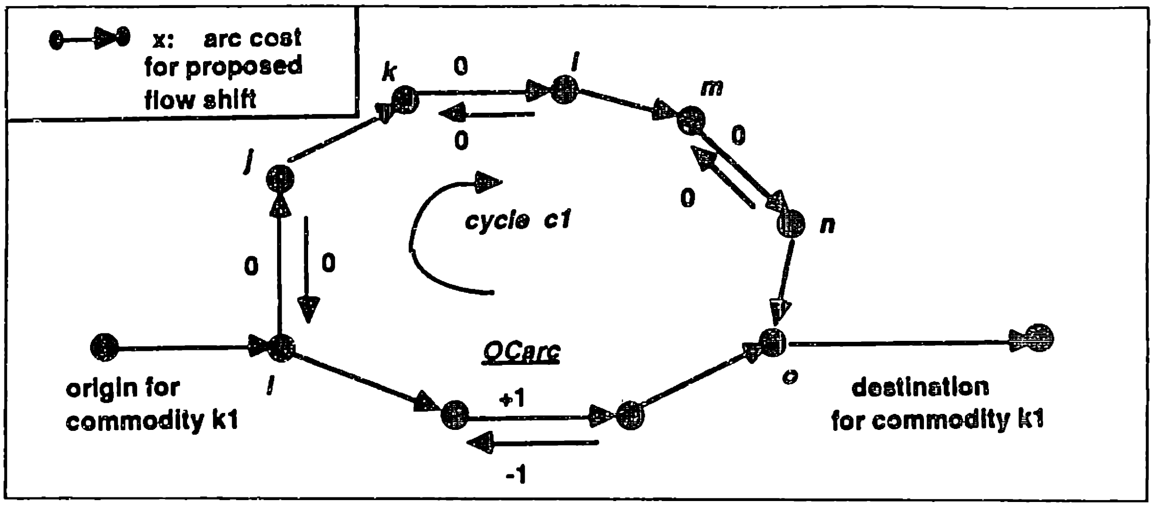
x: arc cost for proposed flow shift
k
0
l
m
0
0
j
0
cycle c1
n
0
0
QCarc
origin for commodity k1
i
+1
o
destination for commodity k1
-1

Figure 4.3a: Cycling in Negative Cost Cycle Search-- Part I

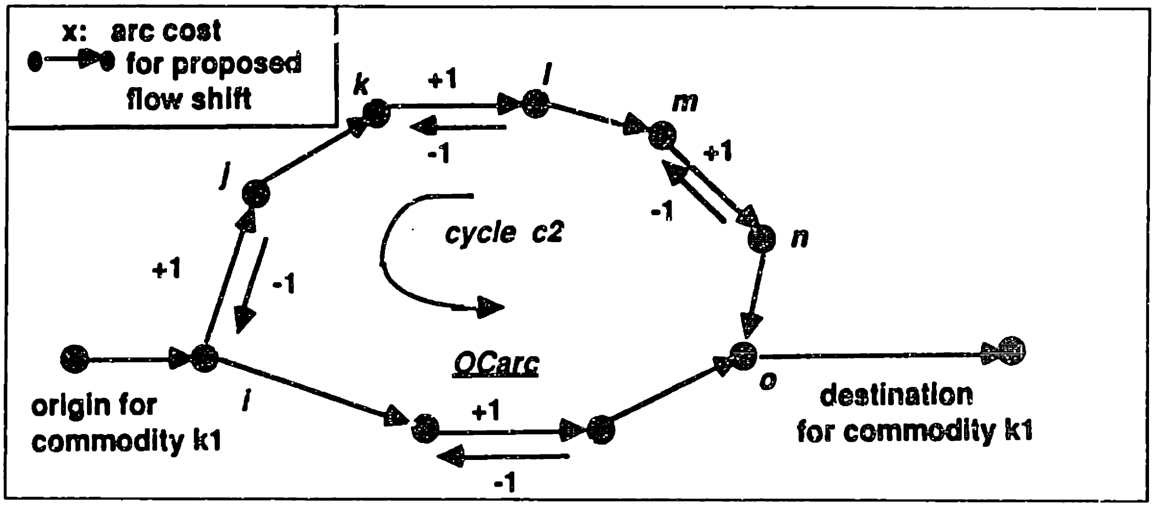
x: arc cost for proposed flow shift
k
+1
l
m
-1
+1
j
-1
cycle c2
n
+1
-1
QCarc
origin for commodity k1
i
+1
o
destination for commodity k1
-1

Figure 4.3b: Cycling in the Negative Cost Cycle Search-- Part II

arcs ij, kl and mn must all be increased from 0 to 1 and the costs of arcs ji, lk and nm must all be decreased from 0 to -1, as shown in Figure 4.3b. A flow shift around cycle c1, as indicated by the revised arc costs, results in an increase, and not a decrease, in infeasibility. The next step then, is to build upon the initial cycle c1 and try to undo some of the "damage" created by shifting flow around c1. Arc ij is selected and a search is made for a negative cost cycle containing arc ji. Note that cycle c2, the reverse of cycle c1, is located as a negative cost cycle. Shifting flow around c2 results in reducing the flow level on arcs ij, kl and mn back to capacity. Hence, to satisfy equations [4.31], the costs on arcs ij, ji, kl, lk, mn and nm should all be set to zero. The result is that the search for a composite negative cost cycle combined flow shifts and adjusted arc costs so as to replicate the original conditions (Figure 4.3a). To control this cycling in the search process, the Flow Adjustment Algorithm (described in Chapter 5) alters arc costs one at a time rather than simultaneously.

<u>4.4.2 Positive Cost Cycles in RN</u>

In the previous section it was shown that because of discrepancies in assigned costs on arcs with flow at capacity, a negative cost cycle does not guarantee that an infeasibility-reducing flow shift is possible. This gives rise to the question of whether an infeasibility-reducing flow shift might be possible around a single, positive cost cycle Γ.

<u>Theorem 4.5:</u> A flow shift around any single positive cost cycle Γ in RN (where arc costs initially are specified in accordance with equations [4.31]), results in an increase in the measure of infeasibility.

Proof: For all arcs with flow not equal to capacity, arc costs in RN can be **fixed** at a particular value to reflect the true change in infeasibility for any proposed composite flow shift containing these arcs. Unfortunately, this same result does not hold for arcs with flow at capacity. Take, for example, an aggregate arc ij^a with flow equal to capacity. Assume either arc ij or arc ji is contained in cycle Γ.

If arc $ij \in \Gamma$, then a proposed flow shift around Γ will over-**capacitate** arc ij^a and hence, the true cost assignment, denoted υ^t_{ij}, should be $\upsilon^t_{ij} = 1$. By equations [4.31], the assigned cost of arc ij cannot exceed the value 1 ($\upsilon_{ij} \le 1$). Thus, for the case where forward arc $ij \in \Gamma$ is over-capacitated by the proposed flow shift around cycle Γ, $\upsilon^t_{ij} \ge \upsilon_{ij}$.

Next consider the case where reverse arc ji is contained in cycle Γ. This time, a proposed flow shift around Γ will **under-capacitate** arc ij^a. Hence, $\Delta^{NET}_{ij^a}$ prior to the flow shift is equal to zero and by equations [4.31], the assigned cost for arc ji is equal to zero, i.e. $\upsilon_{ji} = 0$. If $\pi_{ij^a} = 0$, then $\upsilon^t_{ji} = 0$. Similarly, if $\pi_{ij^a} > 0$, then $\upsilon^t_{ji} = 1$. Thus, for the case where reverse arc $ji \in \Gamma$ and arc ij^a is under-capacitated by the proposed flow shift around cycle Γ, $\upsilon^t_{ji} \ge \upsilon_{ji}$.

Hence for each arc $ij \in \Gamma$ whose corresponding aggregate arc has flow exactly equal to capacity, $\upsilon^t_{ij} \ge \upsilon_{ij}$. Furthermore, for arcs with flow not equal to capacity, arc costs exactly measure the change infeasibility. Thus, it follows that $\upsilon^t_{ij} \geq \upsilon_{ij}$, $\forall ij \in \Gamma$. As a result, the cost of cycle Γ is an underestimate of the change in infeasibility resulting from a flow shift around Γ, i.e. $\Psi_\Gamma \leq \Delta I$. ▨

To summarize, using arc costs defined in equations [4.31], the search for an infeasibility-reducing flow adjustment is carried out by continually updating costs and searching for a set of cycles which when combined produce a composite negative cost cycle. As explained in the next section, the need to search for sets of cycles and to continually update arc costs is attributable to the multi-commodity aspect of the problem.

4.5 Multi-Commodity Effect on Problem Complexity

It is important to realize the effects of the multi-commodity aspect on the problem complexity. Consider the overall flow shift described in Figure 4.2 and assume the problem is a single commodity problem and not a multiple-commodity problem. Then, as shown in Figure 4.4, the sequence of flow shifts depicted in Figure 4.2 can be equivalently represented as a single cycle shift, where conservation of flow is guaranteed for the single commodity and cycle cost $\Psi = -1 = \Delta I$. The single cycle is constructed from the composite set of cycles by eliminating all arcs with a net change in flow of zero.

Thus, for the single-commodity case, a sequence of cycle shifts can always be represented by a single cycle shift in RN. From Theorem 4.3, using arc costs as defined in equation [4.31], the cost of the single cycle is equal to ΔI. Therefore, in the single-commodity case,

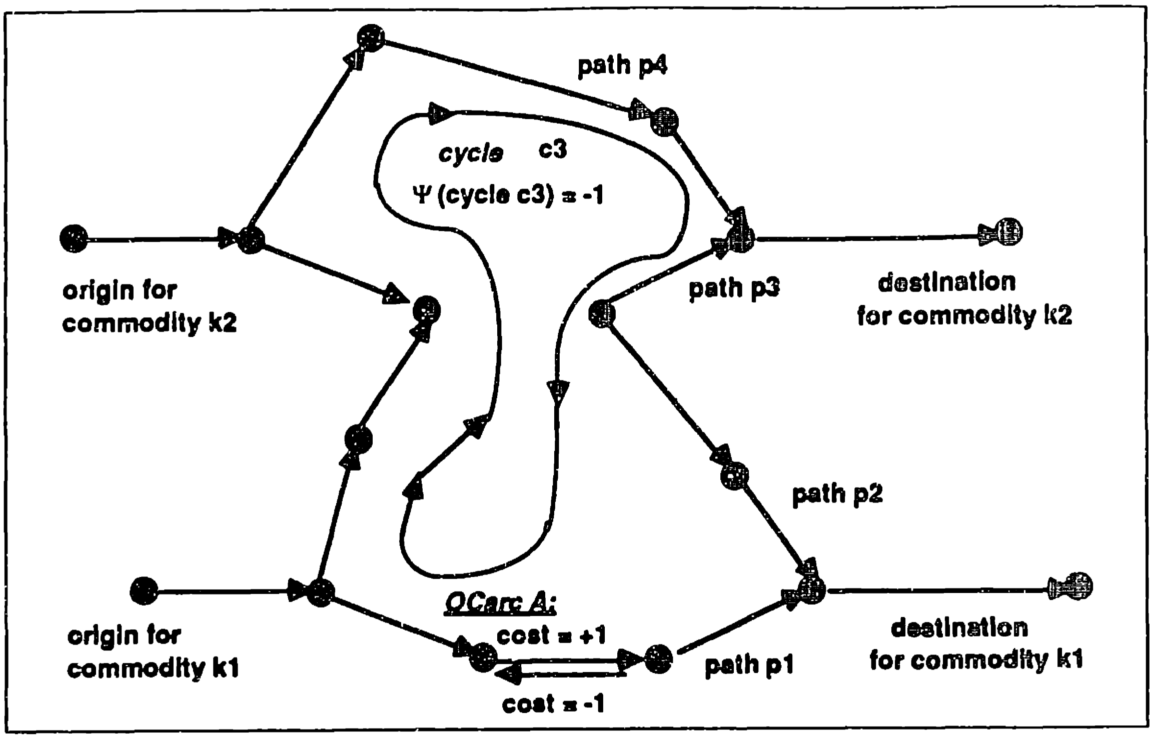

Figure 4.4: Equivalent Single Cycle

the problem is reduced to finding a single, negative cost cycle and shifting flow (in accordance with flow non-negativity constraints) around this cycle. In the multiple-commodity case, this simplification does not hold because the equivalent single cycle for the composite flow shift contains several commodities. For example, cycles cl and c2 (depicted in Figure 4.2) contains one commodity on the arcs contained in paths #1 and #2 and another commodity on the arcs contained in paths #3 and #4. In order to ensure conservation of flow for each commodity, the commodity specific cycle shifts must be individually constructed.

Thus, in a multiple commodity vs. a single-commodity problem, the solution strategy is altered from searching for a single negative cost cycle to searching for multiple commodity-specific cycles which form a composite negative cost cycle.

4.6 Summary

Summarizing, the flow adjustment step is a primal-based step whose purpose is to adjust the current flow assignment by shifting flow between paths in the residual restricted network in a manner which improves the overall condition of problem arcs. The improvement to problem arcs is measured by the change in infeasibility achieved by the flow shift. The search for an infeasibility-reducing flow shift is satisfied by locating a negative cost cycle in RN. However, unlike the single commodity case, this negative cost cycle may need to be a composite cycle containing several individual, commodity-specific cycles. In the search for this negative cost cycle, costs on arcs with flow at capacity must be continually updated to accurately reflect their contribution to the ΔI value. If the flow adjustment step finds a set of commodity-specific cycle shifts which form a negative cost composite cycle, then an appropriately bounded flow shift around the composite cycle is performed, total infeasibility is reduced and the solution is improved. These flow adjustment steps are repeatedly performed until no further reduction in I (or equivalently, in z_{RP}) can be achieved for the given current dual solution.

The next chapter presents the details of the Flow Adjustment Algorithm-- the algorithm developed to locate and execute flow shifts satisfying the PDN algorithmic design features and criterion.

5. The Flow Adjustment Algorithm

The Flow Adjustment Algorithm is used to solve the RP subproblem of the primal-dual method. Given a restricted residual network comprised solely of admissible elements, the objective of the RP problem is to find a flow assignment as close to primal feasibility as possible. This objective is achieved by repeatedly using the Flow Adjustment Algorithm to perform flow adjustment steps meeting the specifications of definition 4.3. The Flow Adjustment Algorithm takes as input the current set of shortest paths and the assignment of flow for each commodity. Then the algorithm searches for a set of commodity-specific flow shifts which form a composite flow shift reducing the infeasibility measure. Equivalently, the Flow Adjustment Algorithm searches for a set of commodity-specific cycles forming a composite negative cost cycle. The algorithm terminates either with an infeasibility-reducing flow shift or with a determination that no further reduction in infeasibility is possible.

This chapter presents the details of the Flow Adjustment Algorithm. Section 5.1 presents an overview with an example demonstration of the algorithm. Section 5 2 then presents the detailed steps of the Flow Adjustment Algorithm. Next, Sections 5.3 through 5.5 discuss termination and finiteness issues. Finally, Section 5.6 summarizes the Flow Adjustment Algorithm.

5.1 Overview and Example

The implementation of the flow adjustment step is guided by the desire to achieve a net decrease in primal infeasibility. To decrease the infeasibility measure, the overall flow shift must achieve a net decrease in the amount by which arcs are over-capacitated or under-capacitated. The idea then is to remove flow from OCarcs or add flow to UCarcs. The flow adjustment step thus begins with the selection of a problem arc (either an OCarc or an UCarc). The algorithm then attempts to improve the condition of this problem arc by searching for a commodity-specific cycle in the restricted, residual network containing the selected arc. If the infeasibility measure can be decreased by shifting flow around this cycle, the flow shift is executed and the flow adjustment step is complete.

In general, the flow adjustment step is not so simple. It is more likely that the located cycle flow shift will increase or hold constant the value of the infeasibility measure. This means that the flow shift, in trying to improve the condition of the selected problem arc, over or under-capacitates at least one other arc. The strategy then is to select the newly created problem arc and repeat the search process. Thus, flow shifts resemble "dominos" in that each flow shift may trigger a whole series of flow shifts. The flow adjustment step constructs these series of flow shifts by repeatedly performing two basic steps:

Step 1) **SELECT** problem arc, where a problem arc is an OCarc, an UCarc or an arc with flow at capacity which becomes an OCarc or an UCarc with a proposed flow shift;

Step 2) **SEARCH** for cycle flow shift improving the condition of the selected problem arc using the commodity-specific restricted, residual network.

The repetition of these select and search steps is halted when the infeasibility measure becomes negative for the set of flow shifts located or when it is established that there is no sequence of flow shifts in RN which improves the infeasibility measure.

The next section provides an example of the **select** and **search** flow adjustment steps and exhibits the "domino-effect".

Example

Consider the network illustrated in Figure 5.1. The flow adjustment step begins with the selection of OCarc A. It locates cycle c1 for commodity k1 assigned to the OCarc. Shifting Δ^* units of k1 around cycle c1 has the positive effect of reducing flow on OCarc A by Δ^* units. Unfortunately, the negative effects of this flow shift outweigh the positive effects. Shifting Δ^* units around cycle c1 has the negative effect of:

i) increasing flow on the capacitated arc B by Δ^* units; and

ii) decreasing the flow on the capacitated arc C (with positive dual price) by an amount Δ^*.

Thus, this flow shift increases the infeasibility measure.

In accordance with the flow adjustment strategy, arc B (an arc whose condition is worsened by flow shift c1) is selected as the new problem arc. Cycle c2 containing arc B is located and a flow shift of

Δ^* around cycle c2 reduces the flow on arc B back to capacity without over or under-capacitating any other arc. Thus, cycle shift c2 negates the ill-effects to arc B caused by cycle shift c1.

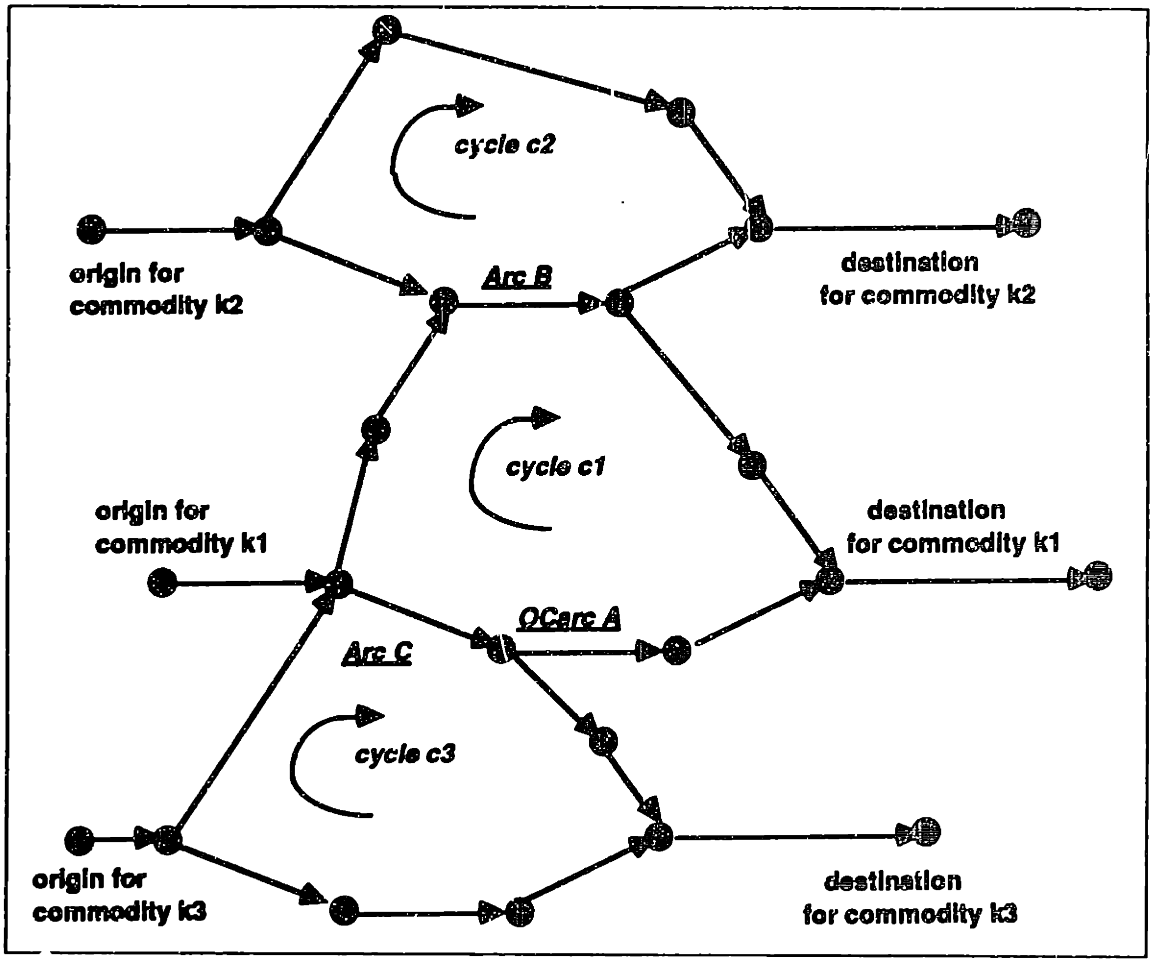

Figure 5.1: Domino-Effect

However, the net change in the infeasibility measure for the composite shift containing cycles c1 and c2 is still non-negative. The net effect of the composite shift is to reduce flow both on the OCarc A and on the capacitated arc C (with positive dual price) by Δ^* units.

Thus, again in accordance with the flow adjustment strategy, the arc C (temporarily transformed into a problem UCarc by the flow shift around cycle c1) is selected as the next problem arc. The search process locates cycle c3 containing UCarc C. Shifting Δ^* units of flow around cycle c3 has the effect of increasing the total flow on arc C back to capacity without over-capacitating or under-capacitating any arc.

By combining cycles c1, c2 and c3 to form a composite shift, flow on the OCarc A is decreased without over-capacitating or under-capacitating any other arcs, thereby reducing the infeasibility measure.

To summarize Figure 5.1, an OCarc is selected and the search for a flow adjustment begins. The domino-effect is experienced and a sequence of flow shifts is generated. Each successive flow shift is centered about a potential problem arc, which if previous flow shifts are executed, becomes either an OCarc or an UCarc. The cycle flow shift alters the flow on the problem arc so as to nullify the illegal effects of the previous flow shifts. Thus, each flow shift "undoes" the "damage" caused by earlier flow shifts. If the flow shift undoes damage at the expense of creating other damage, further flow shifts are triggered until total infeasibility can be strictly reduced.

5.2 Steps of the Flow Adjustment Algorithm

Following is a detailed description of each step in the Flow Adjustment Algorithm, depicted in Figure 5.2.

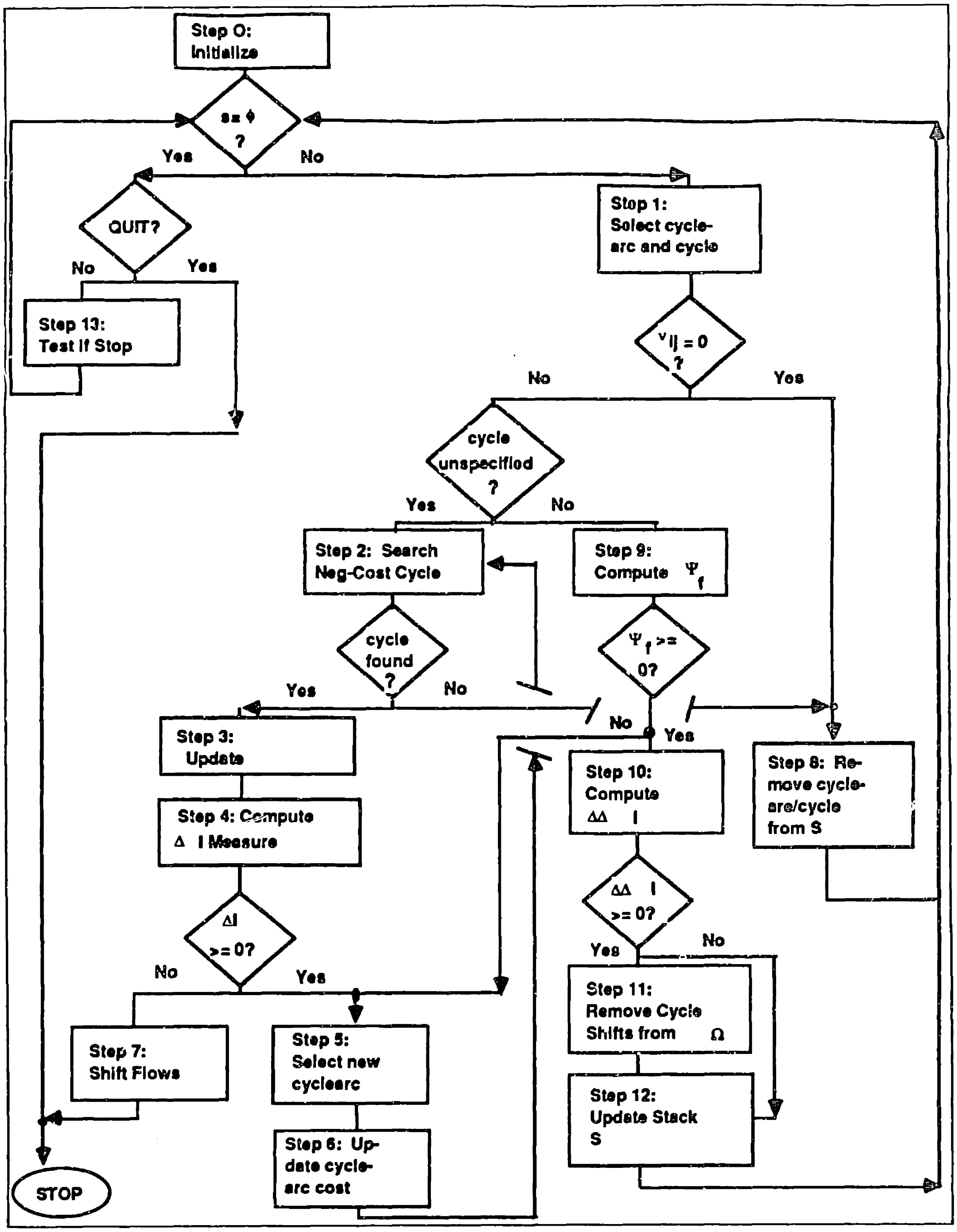

Figure 5.2: The Flow Adjustment Algorithm

5.2.1 FAA Step 0: Initialize

Inputs to the Flow Adjustment Algorithm include a restricted, residual network RN and a set of flow assignments in RN satisfying feasibility in the RP problem (constraints [4.2] through [4.5]).

The first step in the initialization process is to set the arc cost on each arc contained in RN in accordance with equations [4.31].

Next, the initialization step creates a flag called **Quit** and sets the status of this flag to **false**. (The FAA terminates if, when tested, the Quit flag is set to true.) Then, a stack, denoted S, and a cycle list, denoted Ω, are created. Stack S, at any point in the algorithm, contains a listing of arc and cycle pairs and is initialized to contain all ij∈RN with cost strictly greater than zero. The function of Stack S is to keep track of all **unsearched** arcs which have the potential of being members of a negative cost cycle (actually it is negative cost arc ji∈RN which is the member of the negative cost cycle). In addition, arcs are stored in a stack so that their processing order will be last-in-first-out. Prior to the search, each of these arcs is paired with an unlocated and therefore, unspecified cycle.

List Ω, at any point in the algorithm, contains the commodity-specific cycle flow shifts forming the composite flow shift. Initially, List Ω is empty.

5.2.2 FAA Step 1: Select Cyclearc ij and Cycle Γ from Stack S

In order to correct the flow infeasibility on a problem arc, a cycle containing the problem arc must be located and flow shifted around it. The problem arc about which the cycle is formed is referred to as a cyclearc, where a cyclearc is defined as follows:

<u>Definition 5.1:</u> An arc $ij \in RN$ is a **cyclearc** if it satisfies any one of the following criteria:

i) a forward arc $ij \in RN$ with $x_{ij}a > d_{ij}a$, (i.e. an OCarc);

ii) a forward arc $ij \in RN$ with $x_{ij}a = d_{ij}a$ and $\Delta^{NET}{}_{ij}a \geq 0$ for the composite flow shift (i.e. a potential OCarc);

iii) a reverse arc $ij \in RN$ with $x_{ji}a < d_{ji}a$ and $\pi_{ji}a > 0$ (i.e. an UCarc); or

iv) a reverse arc $ij \in RN$ with $x_{ji}a = d_{ji}a$, $\pi_{ji}a > 0$ and $\Delta^{NET}{}_{ji}a \leq 0$ for the current composite flow shift (i.e. a potential UCarc).

Step 1 selects the cyclearc and cycle pair, denoted ij and Γ respectively, from the top of Stack S.

<u>5.2.3 FAA Step 2: Search for Negative Cost Cycle</u>

Step 2 is executed when no cycle has been located for the cyclearc ij selected in Step 1. Thus, in order to improve the status of the problem arc and to improve overall infeasibility, Step 2 searches for a cycle in RN, denoted Γ, which has **negative** cost and contains arc ji. (The search for a negative cost cycle is performed by determining the shortest path in RN from node i to node j. If the cost (length) of this shortest path added to the cost of arc ji is less than zero, a negative cost cycle is found.) Cycle Γ must contain arc ji-- the arc which is opposite and parallel to cyclearc $ij \in RN$, because a flow shift around cycle Γ improves the problem status of the cyclearc by sending flow over its parallel reverse arc. In the case of over-capacitation, the flow shift around Γ sends flow over the reverse arc ji, thereby removing flow from forward arc ij and reducing the amount by which arc ij^a is over-capacitated. Similarly, in the case of under-capacitation, the flow shift around cycle Γ sends flow over the forward arc ji, thereby reducing the amount by which arc ji^a is under-capacitated.

5.2.4 FAA Step 3: Update

Step 3 performs "bookkeeping" tasks. It adds the negative cost cycle Γ located in Step 2 to list Ω-- the current list of commodity-specific cycle flow shifts forming the composite flow shift. In addition, the current cyclearc and cycle pair is updated so that cycle Γ is the one paired with cyclearc ij in stack S.

5.2.5 FAA Step 4: Compute ΔI Measure

Using equations [4.31] (to determine appropriate arc costs) and equations [4.33] and [4.34], Step 4 computes:

i) the change in the infeasibility measure for the composite flow shift, ΔI, and;

ii) the change in the infeasibility measure for an **isolated** flow shift around cycle Γ, denoted ΔI_Γ.

Although the cost of cycle Γ is negative, ($\Psi_\Gamma < 0$), the value of ΔI_Γ is not necessarily negative, as explained in Chapter 4.4.1. Again, this discrepancy arises because costs on arcs with flow exactly at capacity are a function of the assigned arc flows as well as the net change in flow resulting from the composite flow shift. The difficulty is that the value of the net change in flow itself changes as the composite flow shift is constructed. Thus, when cycle Γ is located and added to the composite flow shift Ω, costs on arcs in Γ with flow exactly at capacity may not satisfy equations [4.31].

5.2.6 FAA Step 5: Select New Cyclearc ij from Cycle Γ and Update

By adding cycle Γ to the composite flow shift, the net change in total flow ($\Delta^{NET}{}_{ij}a$) on each arc ij in cycle Γ is altered. For arcs with total flow exactly at capacity (and only these arcs), correct arc costs (equations [4.31]) change as $\Delta^{NET}{}_{ij}a$ values change. Step 5 selects, from the directed cycle Γ in RN, a new cyclearc ij with:

i) total flow equal to capacity; and

ii) arc cost strictly less than the correct arc cost, as determined by equations [4.31], using the $\Delta^{NET}{}_{ij}a$ values computed for the composite flow shift.

As an example, consider forward arc ij$\in$RN contained in cycle Γ with $\upsilon_{ij} = 0$; $X_{ij}a = d_{ij}a$; and $\Delta^{NET}{}_{ij}a > 0$. If the composite flow shift is executed, arc ij^{a}, the potential OCarc, becomes overcapacitated. Thus, in accordance with equations [4.31], the cost of arc ij should be increased to $\upsilon_{ij} = 1$ and the cost of reverse arc ji should be decreased to $\upsilon_{ji} = -1$. Hence, forward arc ij satisfies the selection criteria i) and ii) above and can be chosen as the next cyclearc.

A potential UCarc is another example of an arc meeting the selection criteria of Step 5. Again consider forward arc $ij \in RN$ contained in cycle Γ with cost $\upsilon_{ij} = 0$; $X_{ij}a = d_{ij}a$; $\pi_{ij}a > 0$; and $\Delta^{NET}{}_{ij}a < 0$. If the composite flow shift is executed, arc ij^a becomes undercapacitated. Thus, in accordance with equations [4.31], the cost of arc ij should be decreased to $\upsilon_{ij} = -1$ and the cost of reverse arc ji should be increased to $\upsilon_{ji} = 1$. In this scenario, **reverse arc** ji meets the selection criteria above and can thus be chosen as the next cyclearc.

In summary, Step 5 selects a new cyclearc ij in RN with total flow exactly equal to capacity and an assigned cost which underestimates the arc's contribution to the change in the infeasibility measure for the composite flow shift. Then, cyclearc ij and its paired (yet unspecified) cycle are added to the top of stack S.

<u>Theorem 5.1:</u> There always exists a cyclearc in cycle Γ satisfying selection criteria i) and ii) above.

<u>Proof:</u> Step 5 is executed when, given the current assignment of arc costs, a negative-cost cycle Γ is located (Step 2). Using arc costs as defined by equations [4.31], the cost of cycle Γ (as computed in Step 4) is strictly greater than zero for a flow shift of any size.

Initially, the arc costs of every arc in RN are set in accordance with equations [4.31]. The Flow Adjustment Algorithm alters costs exclusively on arcs with flow exactly equal to capacity. Thus, the difference between the values of Ψ_Γ and ΔI_Γ is attributable to the inappropriate assignment of costs to arcs with total flow equal to capacity. Since Ψ_Γ is strictly less than ΔI_Γ, there exists at least one arc ij with flow equal to capacity and arc cost strictly less than the correct arc cost as specified by equations [4.31]. ■

Theorem 5.1 reaffirms the domino phenomenon. If $\Delta I \geq 0$, then the proposed flow shift, in correcting the infeasible flow assignment on some arc, over or under-capacitates at least one other arc. Thus, the algorithm proceeds by selecting one of these arcs and tries to correct its newly infeasible flow assignment.

5.2.7 FAA Step 6: Update Cyclearc Cost

To more accurately measure the change in the infeasibility measure resulting from the composite flow shift, Step 6 **updates** the cost of the cyclearc ij selected in Step 5. The cyclearc cost, υ_{ij}, is altered as follows:

$$\Delta\upsilon_{ij} = \mathrm{MIN}(-\Psi_\Gamma,\ \upsilon^*_{ij} - \upsilon_{ij}) \qquad [5.1]$$

where:

$\Delta\upsilon_{ij}$: amount **added** to the cost of arc ij∈RN and the amount **subtracted** from the cost of its opposing, parallel arc ji∈RN;

Ψ_Γ: cost of cycle Γ before cost of cyclearc ij is updated;

υ^*_{ij}: cost of arc ij as determined by equations [4.31] given current composite flow shift;

υ_{ij}: current assigned cost of arc ij.

Figure 5.2 shows that Step 6 is executed only if the cost of cycle Γ is strictly negative and thus, $-\Psi_{\Gamma} > 0$. Furthermore Step 5, by definition, selects cyclearc ij only if its assigned cost is strictly less than the true arc cost as determined by equations [4.31]. Thus, it follows that $\upsilon^{*}_{ij} - \upsilon_{ij} > 0$. Hence, $\Delta\upsilon_{ij}$, as determined by equation [5.1], is the minimum of two strictly positive values and is therefore strictly positive. As a result, the cost of cyclearc $ij \in RN$ **strictly** increases in Step 6, i.e. $\Delta\upsilon_{ij} > 0$.

The updated cost for arc ij is achieved by **adding** $\Delta\upsilon_{ij}$ to its current arc cost. Similarly, since arc costs on parallel forward and reverse arcs in RN are negative images of one another, the updated cost for the opposing parallel arc $ji \in RN$ is achieved by **subtracting** $\Delta\upsilon_{ij}$ from its current assigned cost.

A strict increase in the cost of cyclearc $ij \in \Gamma$ results in a strict increase in the cost of cycle Γ, which is a negative cost cycle. Thus, in addition to more accurately reflecting the contribution from arc ij to the change in the infeasibility measure for a composite flow shift containing Γ, the cost adjustment works to eliminate the negative-cost cycle. The size of the cost adjustment to arc ij (equation 5.1) is bounded, however, so that the cost of cycle Γ does not become positive. Increasing the cost of cycle Γ above zero would be counter-productive since then, traversing Γ in the reverse direction would yield a cycle of negative cost.

Altering costs in the above manner is consistent with the economic interpretation presented in section 2.2.3 for the Primal-Dual Method. If arc ij^a is a potential OCarc, forward arc $ij \in RN$ is selected as the cyclearc. On the other hand, if arc ij^a is a potential UCarc, reverse arc $ji \in RN$ is the selected cyclearc. In either case, the cost of the cyclearc is always increased. The increased cost makes the cyclearc less attractive and its opposing parallel arc more attractive. As a result, removal of flow is encouraged for (potential) OCarcs and addition of flow is encouraged for (potential) UCarcs.

5.2.8 FAA Step 7: Shift Flows

The Flow Adjustment Algorithm shifts flow either when $\Delta I < 0$ or when $\Delta I_\Gamma < 0$. The goal is to achieve a maximum decrease in the infeasibility measure given that an equal quantity of flow, denoted Δ^*, is shifted around each cycle in the flow shift. (Note: If it is desirable to shift more flow around some cycle, then that cycle may be contained in the composite flow shift more than once.) The quantity Δ^* is restricted in size by the flow non-negativity constraints and the requirement that the flow shift reduce the infeasibility measure.

To satisfy the flow non-negativity constraints, the amount of flow shifted off an arc can not exceed the quantity of that commodity assigned to the arc, i.e.:

$$\Delta_x^* = \mathrm{MIN}_{\Gamma \in \Omega}\ \mathrm{MIN}_{ji \in F^c \cap \Gamma}\ (x_{ij}a^k,\ k \text{ shifted around } \Gamma) \qquad [5.2]$$

where:

Δ_x^*: optimal amount of flow to shift given flow non-negativity requirements only.

Δ^* must also be bounded to ensure that the infeasibility measure is strictly decreased by the composite flow shift. Equations [4.31] appropriately measure the change in infeasibility for a composite flow shift provided that the amount of flow shifted does not push the flow level of any arc across its capacity boundary. In other words, Theorem 4.4 holds if the flow shift does not change an arc with less flow than capacity into an OCarc, or alternatively, does not change an OCarc into an arc with less flow than capacity. These restrictions are expressed as follows:

$$\Delta_I^* \leq \mathrm{MIN}_{\Omega^1 \cup \Omega^2} \left((d_{ij}a - X_{ij}a)/\Delta^{NET}{}_{ij}a\right) \qquad [5.3]$$

where:

Δ_I^*: amount of flow which can be shifted before at least one arc cost no longer correctly measures its contribution to ΔI for the composite flow shift;

Ω^1: set of forward arcs $ij \in RN$ where $X_{ij}a < d_{ij}a$, $\Delta^{NET}{}_{ij}a > 0$;

Ω^2: set of reverse arcs $ji \in RN$ where $X_{ij}a > d_{ij}a$, $\Delta^{NET}{}_{ij}a < 0$.

Using equations [5.2] and [5.3], flow non-negativity and a reduction in the infeasibility measure is guaranteed for a flow shift of size Δ^*, where Δ^* is defined as follows:

$$\Delta^* = \mathrm{MIN}(\Delta_X^*, \Delta_I^*). \qquad [5.4]$$

Given the above definition of Δ^*, the following observations are made:

<u>Observation 5.1:</u> In the PDN algorithm, non-integer flow shifts may occur. For example, when an arc ij is contained in more than one cycle of the composite flow shift, $\Delta^{NET}_{ij}a$ may be strictly greater than or less than one and consequently, Δ_I^* will be a fractional value.

<u>Observation 5.2:</u> Each flow adjustment step terminating with a flow shift will **strictly** decrease the infeasibility measure. By design of the PDN algorithm, a flow shift is attempted only when an infeasibility reducing flow shift of size Δ^* is located. Δ^* is guaranteed to be a non-zero quantity because it is bounded only by the requirements that:

i) OCarcs are not transformed into UCarcs; and analogously

ii) UCarcs are not transformed into OCarcs.

Thus, each flow shift **strictly** reduces the infeasibility measure.

Equation [5.4] provides a **lower bound** on the total amount of flow which can be shifted to achieve the maximum reduction in the infeasibility measure. If $\Delta^* < \Delta_X^*$, then the overall value of ΔI may still be negative for shifts of flow in excess of the Δ^* amount, even though the value of ΔI is increased. Hence, the determination of the optimal flow shift size is an iterative process, as shown in Figure 5.3.

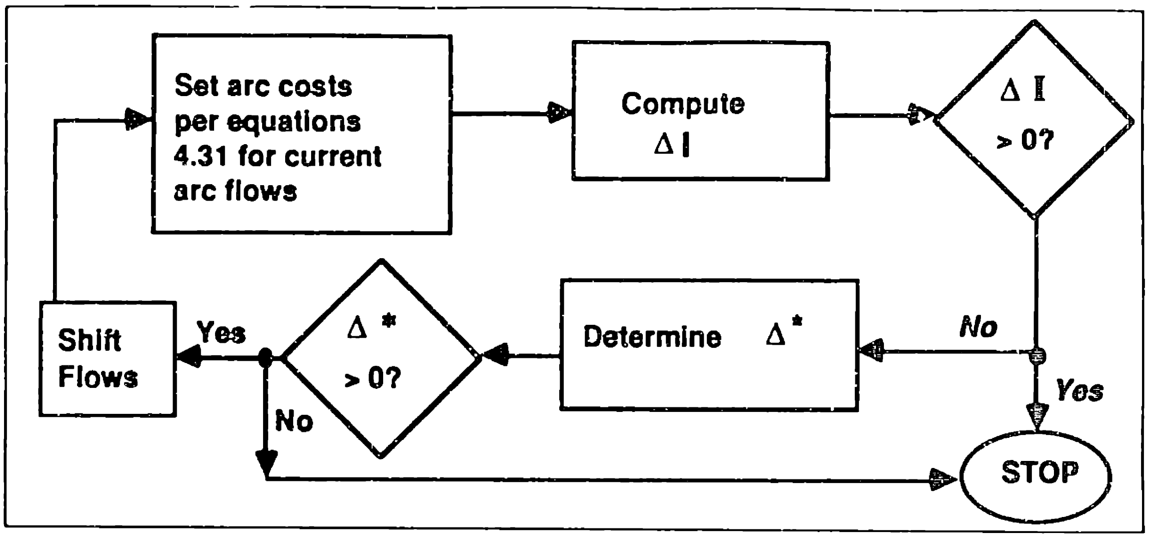

Figure 5.3: Determination of the Optimal Flow Shift Size

Using equation [5.4], the "conservative" flow shift size is computed, then Δ^* units of flow are shifted around each cycle $\Gamma \in \Omega$. The arc flow levels are adjusted to reflect the flow shift of Δ^* units and the arc costs are updated with equations [4.31]. The value of ΔI is again computed using equations [4.33] and [4.34]. If ΔI is still negative, another flow shift will strictly reduce infeasibility. Thus, the flow quantity Δ^* is again determined, a flow shift of Δ^* units is executed and the entire process repeats. The optimal flow shift is achieved when either $\Delta^* = \Delta_x{}^*$ or infeasibility is not decreased for any further flow shift.

If step 7 is executed, existing commodity flows are shifted between shortest paths and a net decrease in the total amount of primal infeasibility is achieved. Thus, the Flow Adjustment Algorithm successfully locates a flow shift satisfying all of the design features and criterion of the PDN algorithm.

5.2.9 FAA Step 8: Remove Cyclearc ij and Cycle Γ Pair from Stack S

Step 8 shown in Figure 5.2 is executed when the cost of arc ij is equal to zero or no negative cost cycle is associated with the arc ij. In either case, the search for a negative cost cycle for arc ij is considered complete and cyclearc ij and its paired cycle Γ are removed from the top of stack S. (If arc ij has cost equal to zero and a negative cost cycle containing it exists, then there must be at least one other arc with non-zero cost contained in the cycle. When that other arc is selected and a search is launched, the negative cost cycle will be found.)

5.2.10 FAA Step 9: Compute Ψ_Γ

Step 9 is executed when the cyclearc popped off Stack S is paired with a previously located cycle Γ. When located, cycle Γ had negative cost. However, subsequent adjustments to arc costs (Step 6) may have altered the cycle cost. Thus, the cost of cycle Γ, denoted Ψ_Γ, is computed using equation [4.34] and the currently assigned arc costs.

5.2.11: FAA Step 10: Compute Change in ΔI due to Cycle Γ

Equation [4.33] is used to compute the value of the change in infeasibility associated with:

i) the composite flow shift containing cycle Γ; and

ii) the composite flow shift not containing cycle Γ.

Then, the change in ΔI caused by adding cycle Γ to the composite flow shift is computed as follows:

$$\Delta(\Delta I) = \Delta I(\Gamma) - \Delta I(\backslash\Gamma) \tag{5.5}$$

where:

$\Delta(\Delta I)$: change in the ΔI value realized when cycle Γ is added to the composite flow shift;

$\Delta I(\Gamma)$: the value of ΔI achieved for the composite flow shift containing cycle Γ;

$\Delta I(\backslash\Gamma)$: the value of ΔI achieved for the composite flow shift not containing cycle Γ.

Even though cycle Γ has positive cost and Theorem 4.5 states that a flow shift around a **single** positive cost cycle results in $\Delta I > 0$, it is possible that $\Delta(\Delta I)$ is less than zero for the **composite** cycle case. Thus, the computations of this step are necessary.

5.2.12 FAA Step 11: Remove Cycle Shifts from List Ω

Step 11 is executed when the value of $\Delta(\Delta I)$ (as computed in step 10) is strictly positive. This means that in adding cycle Γ to the composite cycle, infeasibility is increased. Since the objective is to construct a composite negative cost cycle, cycle Γ (which has **non-negative** cost) should not be included in the composite cycle. Thus, cycle Γ is removed from list Ω, thereby removing it from the composite flow shift.

The original inclusion of cycle shift Γ in the composite flow shift may trigger a succession of flow shifts, each one undoing an ill-effect of the previous one. It is not necessary, or reasonable, to include in the composite flow shift these flow shifts triggered by the now eliminated cycle shift Γ. Therefore, along with cycle Γ, Step 11 removes from list Ω each of the cycle shifts triggered by cycle shift Γ. Special data structures are used to facilitate the removal of all appropriate cycles and cyclearcs from list Ω and stack S.

5.2.13 FAA Step 12: Update Stack S

Step 12 changes the cycle in the cyclearc ij and cycle Γ pair to an **unspecified** (not yet located) cycle, thus modifying, not removing, the cyclearc and cycle pair in Stack S. This allows another search to be performed for an alternative negative cost cycle containing arc ji.

5.2.14 FAA Step 13: Test if Terminate

Step 13 is executed when Stack S is empty. In this case, a search for negative cost cycles has been performed for all arcs with negative cost. However, subsequent to the search, adjustments to arc costs (Step 6) may result in the creation of negative cost cycles. Thus, Step 13 checks to ensure that, indeed, all negative cost cycles in RN have been eliminated. This test is performed by first initializing stack S to contain all arcs ij∈RN with positive cost. For each arc in stack S, a search for a negative cost cycle (step 2) is performed. If no negative cost cycle exists, the arc is removed from Stack S. However, if a negative cost cycle does exist and the arc has flow not equal to capacity, the arc and negative cost cycle pair remain in Stack S. If a negative cost cycle does exist but the arc has flow exactly equal to capacity, its cost and the cost of the parallel, opposing arc are set to zero and the arc is removed from Stack S. This ensures that Stack S contains only problem arcs with primal infeasible flows which can be corrected through appropriate flow shifts. There is no motivation to locate a flow shift centered about an arc with flow exactly equal to capacity because:

i) its flow assignment is not primal infeasible; and

ii) a flow shift is likely to transform the feasible flow assignment into an infeasible flow assignment.

If after testing each arc, Stack S is empty, the quit flag is set to true and the algorithm terminates. Otherwise, Stack S has new elements and the algorithm reiterates beginning with Step 1.

Thus, the following proposition holds by design of the Flow Adjustment Algorithm:

Proposition 5.1: If Stack S is empty and the Quit flag status is equal to true, there are no negative cost cycles in RN. ▪

5.3 Termination of the Flow Adjustment Algorithm

If an infeasibility-reducing flow shift is located, or Stack S is empty, the Flow Adjustment Algorithm of Figure 5.2 terminates. Termination of the algorithm with a flow shift implies that infeasibility is reduced with an appropriately bounded flow shift around a negative cost composite cycle (Observation 5.2). From Theorem 4.2, a reduction in primal infeasibility can be interpreted as advancement towards optimality in the CMCF problem, P. Thus, the flow shift of step 7 represents a successful execution of the primal-based Flow Adjustment Step. Existing commodity flows are shifted between shortest paths such that total primal infeasibility is reduced and advancement towards optimality is achieved.

The Flow Adjustment Algorithm can also terminate when Stack S is empty. In this case, no flow shift is executed and consequently, the flow adjustment step fails in its quest to move the primal solution closer to feasibility. However, this so-called failed flow adjustment step is successful in optimally solving the RP/DRP problem, as demonstrated in the following proofs. First it is shown that the primal-based Flow Adjustment Algorithm generates dual feasible solutions to the DRP problem. Then, it is shown that the dual and primal solutions existing at the termination of the failed flow adjustment step optimally solve the RP/DRP problem.

<u>Theorem 5.2:</u> Using the arc costs generated in the failed flow adjustment step, a feasible solution to the DRP problem can be easily constructed.

<u>Proof:</u> The flow adjustment algorithm fixes the costs on OCarcs and UCarcs in accordance with equations [4.31] and then adjusts costs only on arcs with flow exactly at capacity. The size of the cost adjustment for arcs with flow at capacity is bounded (equation [5.1])

such that the cost of the arc in RN is ≤ 1; is ≥ 0 for arcs with $\pi = 0$; and is ≥ -1 for arcs $\pi \neq 0$.

Let υ_{ij} be the cost assigned arc ij when the failed flow adjustment step terminates and let $\Delta\pi_{ij}$ equal υ_{ij}, $\forall ij \in RN$. Then, the values of $\Delta\pi_{ij}$ satisfy constraints [2.23] and [2.24] in the DRP problem formulation. Furthermore, using arc costs of $\Delta\pi_{ij}$ and restricting the analysis to the set of shortest paths only, let $\Delta\sigma^k$ represent the cost (i.e. length) of the shortest path for commodity k. It follows then that for all $k \in K$:

$$\Delta\sigma^k \leq \Sigma_{ij \in RN}\, \Delta\pi_{ij}\delta_{ij}{}^{p,k}, \quad \forall p \in J(k) \qquad [5.6]$$

Equation [5.6] shows that $\Delta\sigma^k$, $\forall k \in K$, as defined above, satisfy constraints [2.22] of the DRP problem. Hence, using the solution to the failed flow adjustment step, a feasible solution to the DRP problem is generated by setting $\Delta\pi_{ij} = \upsilon_{ij}$, $\forall ij \in RN$ and letting $\Delta\sigma^k =$ shortest path length in RN, $\forall k \in K$. ∎

The above definitions of $\Delta\pi$ and $\Delta\sigma$ lead to the following theorem:

<u>Theorem 5.3</u> For each commodity, all flow carrying paths have minimum length, i.e.:

$$\Delta\sigma^k = \Sigma_{ij \in RN}\, \Delta\pi_{ij}\delta_{ij}{}^{p,k}, \quad \forall p \in J(k) \cap JX(k) \qquad [5.7]$$

where:

JX(k): set of all paths with flow of commodity k.

<u>Proof:</u> Proving by contradiction, assume that some path p^k

carrying flow of commodity k has length greater than the shortest path, denoted p^{*k}, for commodity k. At least one non-zero cost cycle is formed by paths p^k and p^{*k} since the paths are of unequal length and have at least two nodes in common, namely the origin and destination node for commodity k. Call the non-zero cost cycle Γ and compute its cost by traversing the arcs on path p^k in the reverse direction (all these arcs exist because path p^k has flow of commodity k) and traversing the arcs on path p^{*k} in the forward direction. Then, since path p^{*k} is the shortest path and p^k is longer, the cycle has negative cost. This is a contradiction however, because by Proposition 5.1, when the flow adjustment step terminates with stack S empty, all negative cost cycles in RN have been eliminated. ∎

<u>Theorem 5.4:</u> The primal and dual solutions existing at the termination of the failed Flow Adjustment Step optimally solve the RP and DRP problems, respectively.

<u>Proof:</u> The initialization step of the Flow Adjustment Algorithm (Step 0) begins with a flow assignment satisfying feasibility for the RP problem and then proceeds to shift flow around cycles such that flow non-negativity and conservation of flow are preserved. Thus, the Flow Adjustment Algorithm maintains primal feasibility in RP throughout and therefore, terminates with a solution feasible to the RP problem.

Second, as shown in Theorem 5.2, when the failed Flow Adjustment Algorithm terminates, a feasible solution to the DRP problem is constructed by setting $\Delta\pi_{ij} = \upsilon_{ij}$, $\forall ij \in A$ and $\Delta\sigma^k$ = shortest path length for commodity k in RN, $\forall k \in RN$.

Finally, the complementary slackness conditions for the RP/DRP problem are as follows:

#1a: $-\Delta\pi_{ij}(\Sigma_{k\in K} \Sigma_{p\in P^k} x_p{}^k \delta_{ij}{}^{p,k} + w_{ij}{}^+ - w_{ij}{}^- - d_{ij}) = 0$, $\forall ij \in IJ^+(\pi)$; [5.8a]

#1b: $-\Delta\pi_{ij}(\Sigma_{k\in K} \Sigma_{p\in P^k} x_p{}^k \delta_{ij}{}^{p,k} - w_{ij}{}^- + s_{ij} - d_{ij}) = 0$, $\forall ij \in IJ^+(\pi)^c$; [5.8b]

#2: $\Delta\sigma^k(\Sigma_{p\in P^k} x_p{}^k - b^k) = 0$, $\forall\ k \in K$; [5.9]

#3: $x_p{}^k(-\Sigma_{ij\in A} \Delta\pi_{ij}\delta_{ij}{}^{p,k} - \Delta\sigma^k) = 0$, $\forall p \in P^k$, $\forall k \in K$; [5.10]

#4a: $(\Delta\pi_{ij}-1)(w_{ij}{}^-) = 0$, $\forall\ ij \in A$. [5.11a]

#4b: $(\Delta\pi_{ij}+1)(w_{ij}{}^+) = 0$, $\forall\ ij \in IJ^+(\pi)$ [5.11b]

#4c: $\Delta\pi_{ij}(s_{ij}) = 0$, $\forall\ ij \in IJ^+(\pi)^c$ [5.11c]

The primal and dual solutions existing at the termination of the failed Flow Adjustment Step satisfy:

i) complementary slackness conditions #1 and #2 by feasibility in RP of the primal solution;

ii) complementary slackness conditions #3 by Theorem 5.3; and

iii) complementary slackness conditions #4 by equations [4.31] together with the Flow Adjustment Algorithm which fixes the cost (= $\Delta\pi_{ij}$) on over-capacitated arcs (i.e. arcs with $w_{ij}{}^- > 0$) to +1; the cost on under-capacitated arcs (i.e. arcs with $w_{ij}{}^+ > 0$) to -1; and all other arcs with flow strictly less than capacity (i.e. arcs with

$s_{ij} > 0$) to 0. Furthermore, as shown in Section 2.4 of Chapter 2, it is possible only for the above defined arcs to have positive values for w_{ij}^+, w_{ij}^- and s_{ij}.

Thus, the primal and dual solutions existing at the termination of the failed Flow Adjustment Step satisfy primal feasibility, dual feasibility and complementary slackness conditions and therefore, optimally solve the RP and DRP problems, respectively. ■

The above results will be used in Chapter 6 to show how the arc costs generated in the failed flow adjustment step are used to alter the dual solution in the Price Adjustment Step.

5.4 Finiteness and Flow Adjustment Steps

Although the CMCF problem is a network flow problem, the integrality property (which states that the solution is integer if upper and lower bounds on variables and right-hand side values are integer) does not hold because of the bundle constraints (equations [2.2]). As a result, the optimal solution may not contain integer flow assignments to paths. This implies that the quantity of flow shifted in a Flow Adjustment Step need not be integer and thus, the decrease in infeasibility attained for each flow shift may be non-integral. Hence, the number of flow adjustment steps necessary to solve the RP/DRP problem may not be finite.

Consider the example depicted in Figures 5.4. First, Figure 5.4a shows the initial assignment of flows input into the Flow Adjustment Algorithm. Arc a1 is over-capacitated by one unit and all other arcs

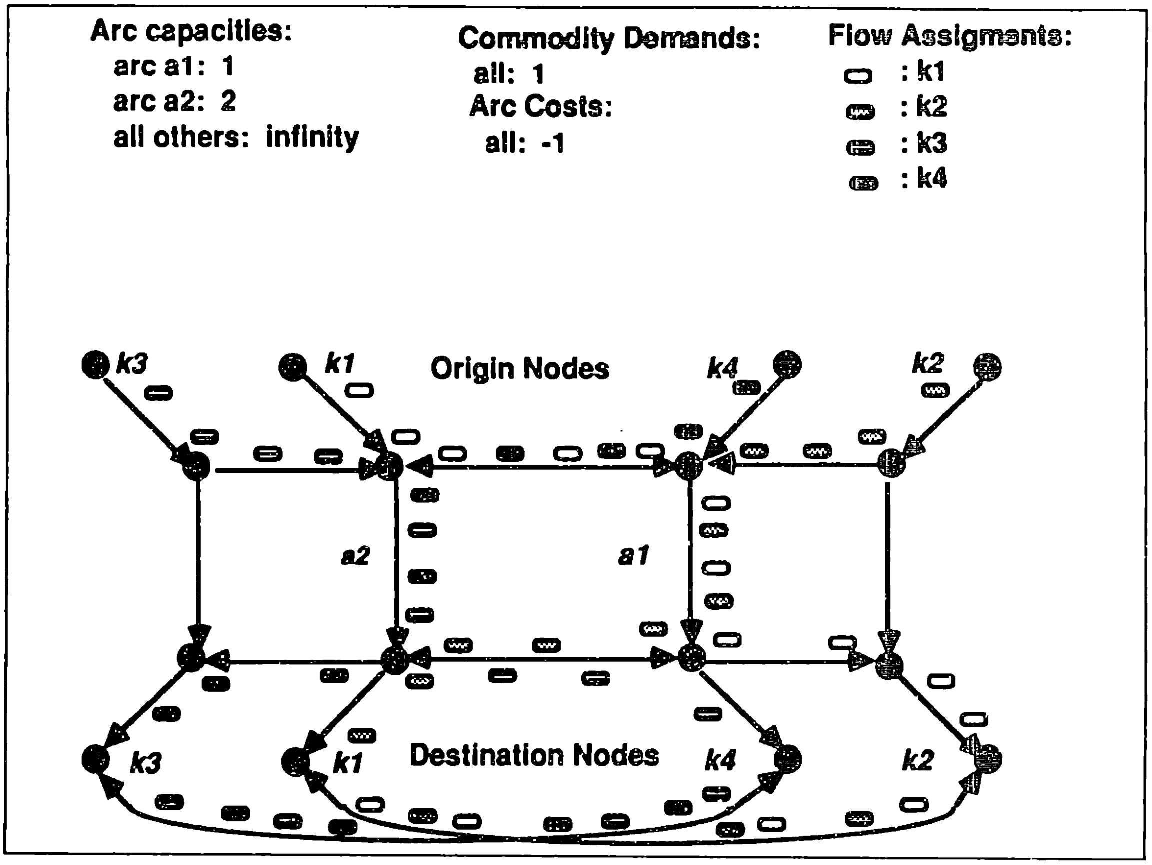

Figure 5.4a: Infinite Sequence of Flow Adjustment Steps-- Part I

have primal feasible flow assignments. Figure 5.4b shows the flow shift located by the Flow Adjustment Algorithm. Using equations [5.2], [5.3] and [5.4], the size of this flow shift is determined to be 1/2 unit. In executing the flow shift, one unit of flow is removed from OCarc a1, arc a2 is over-capacitated by 1/2 unit and all of the other arc flows remain primal feasible ($\Delta I = -1/2$). The resulting assignment of flows is shown in Figure 5.4c. Next, arc a2 is selected as the problem arc and the flow shift of Figure 5.4d is located. The size of this flow shift is determined to be 1/4 unit. In executing

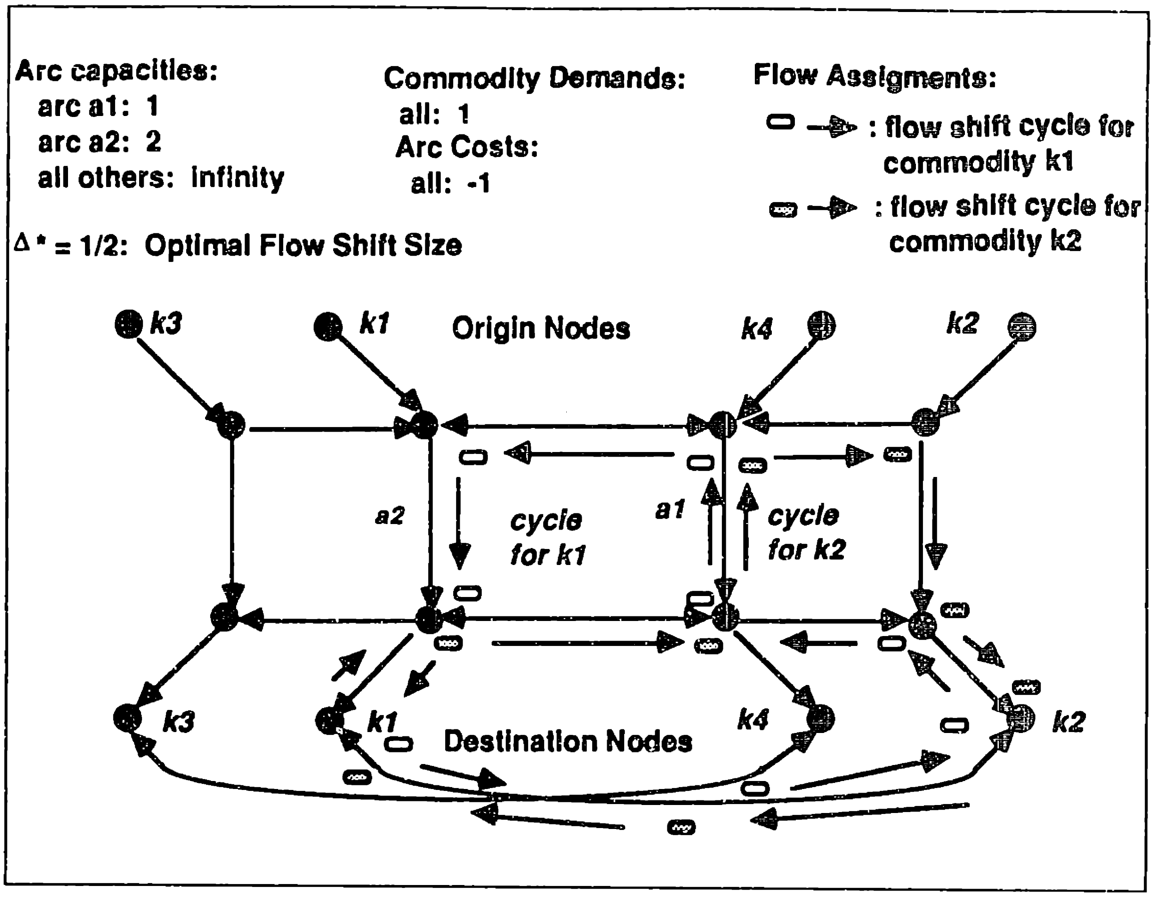

Figure 5.4b: Infinite Sequence of Flow Adjustment Steps-- Part II

the flow shift, 1/2 unit of flow is removed from OCarc a2, arc a1 is over-capacitated by 1/4 unit and all other arc flows remain primal feasible (ΔI = -1/4). The Flow Adjustment Algorithm continues by selecting OCarc a1 and again, locating the infeasibility reducing flow shift shown in Figure 5.4b. This time however, the size of the flow shift is reduced to 1/8. Notice the emerging pattern. The flow shifts of Figure 5.4b and 5.4d, although decreasing in size, can be repeatedly performed to achieve a decrease in infeasibility.

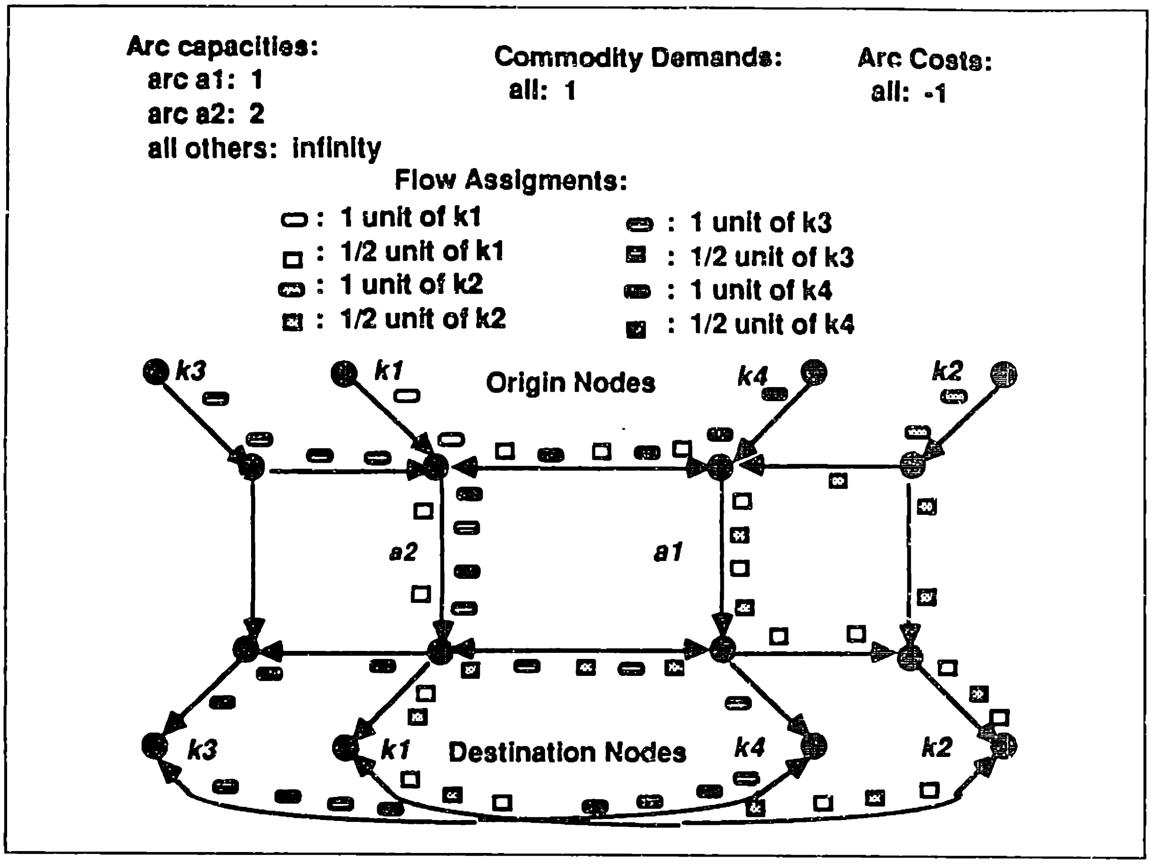

Figure 5.4c: Infinite Sequence of Flow Adjustment Steps-- Part III

The size of the ith flow shift, denoted S^i, is expressed as:

$$S^i = 1/(2^i). \qquad [5.12]$$

Thus, to reduce infeasibility to zero, an infinite sequence of flow shifts must be executed. To ensure that an infinite number of flow adjustment steps does not occur in solving the RP/DRP problem, finiteness is guaranteed by employing the simplex algorithm.

When this "unacceptable" advancement towards feasibility is achieved for a specified number of flow adjustment steps, the simplex

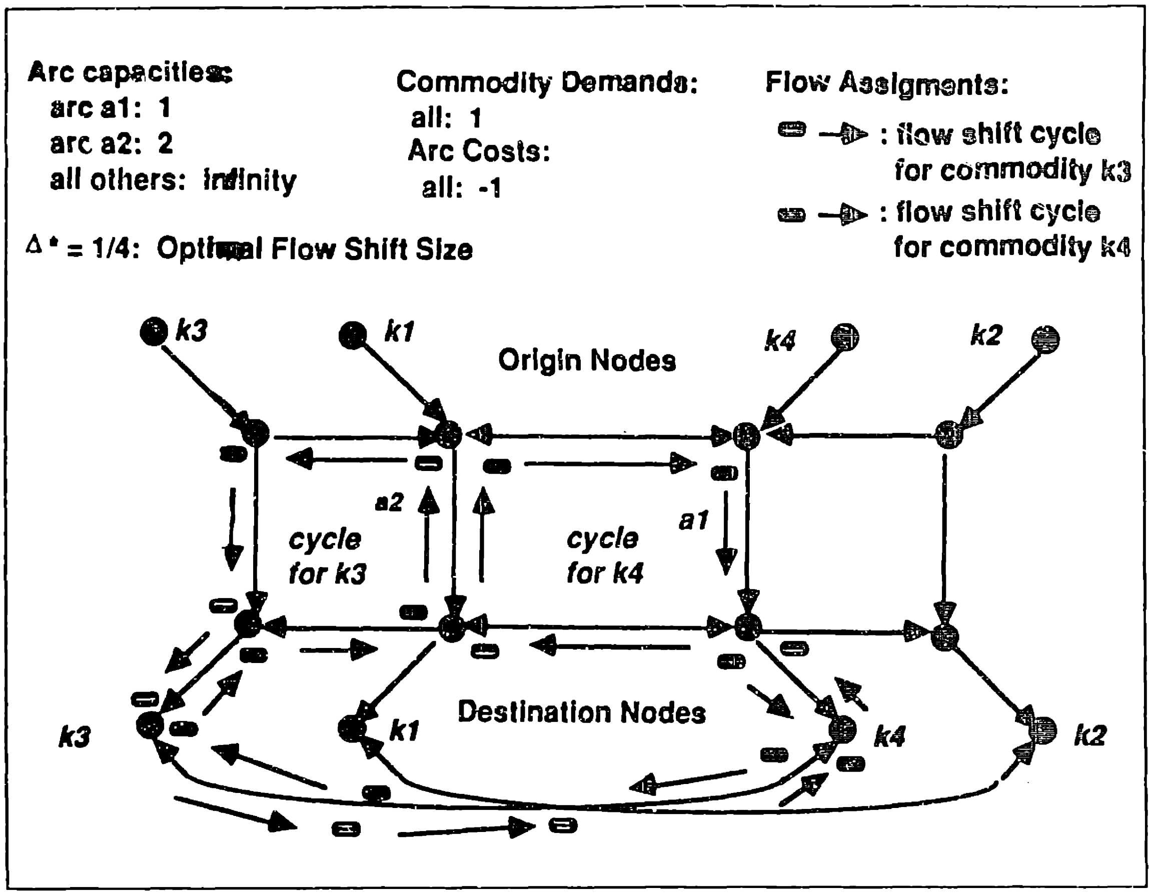

Figure 5.4d: Infinite Sequence of Flow Adjustment Steps-- Part IV

method is used to solve either the RP or the DRP problem formulation. The DRP problem may be favored over the RP problem since DRP has potentially far fewer rows than does RP. Given the path formulation of the CMCF problem, the number of rows in the RP problem formulation is equal to the number of capacitated arcs plus the number of commodities. In cases where the capacitated network is very large, the size of the RP problem is excessive. In the DRP problem formulation, however, the number of rows is equal to the number of utilized shortest paths for each commodity. Thus, depending of course on the nature of the problem, the simplex method may be able to accommodate the DRP problem size, even in cases where the RP problem size is unmanageable.

The question which then arises is why the RP/DRP problems at each iteration of the PDN algorithm cannot be solved by applying the simplex method to the DRP problem formulation. The answer lies in the issue of efficiency. In solving the DRP problem, the optimal solution to the i^{th} iteration DRP problem is not a feasible solution to the $i+1^{st}$ iteration DRP problem with its additional constraints. (In the case of the RP problem, the optimal solution at the i^{th} iteration is always a feasible starting solution to the $i+1^{st}$ RP problem with its additional variables.) Thus, using the simplex method to solve the DRP problem at each iteration of the Primal-Dual method would be inefficient. A good starting solution is not readily available and each LP must be solved starting with Phase I.

If the simplex algorithm is unable to solve either the DRP or the RP problem (because of size limitations), the PDN algorithm is not guaranteed to produce the optimal solution to the CMCF problem. It can, however, advance towards feasibility using a different starting cyclearc and, in the end, may achieve optimality or near-optimality in the CMCF problem. A measure of the closeness of the final solution to optimality can be determined using a heuristic to find a feasible flow assignment in P. This feasible flow assignment is constructed by altering the solution existing when the Flow Adjustment Algorithm unsuccessfully terminates. The objective function value of this primal feasible solution in P is an upper bound on the optimal objective function value, z^*_P. The lower bound on z^*_P is obtained by calculating the value of z_D corresponding to the current (always) feasible solution in D existing at the termination of the Flow Adjustment Algorithm.

5.5 Cycling and Flow Adjustment Steps

Another problem with the Flow Adjustment Algorithm with respect to finiteness involves cycling in the search for an infeasibility-reducing flow shift. Consider the example shown in Figures 5.5. Arc a1 is over-capacitated and thus, cycle shift 1 is located by the Flow Adjustment Algorithm (Figure 5.5a). Cycle shift 1 removes flow from

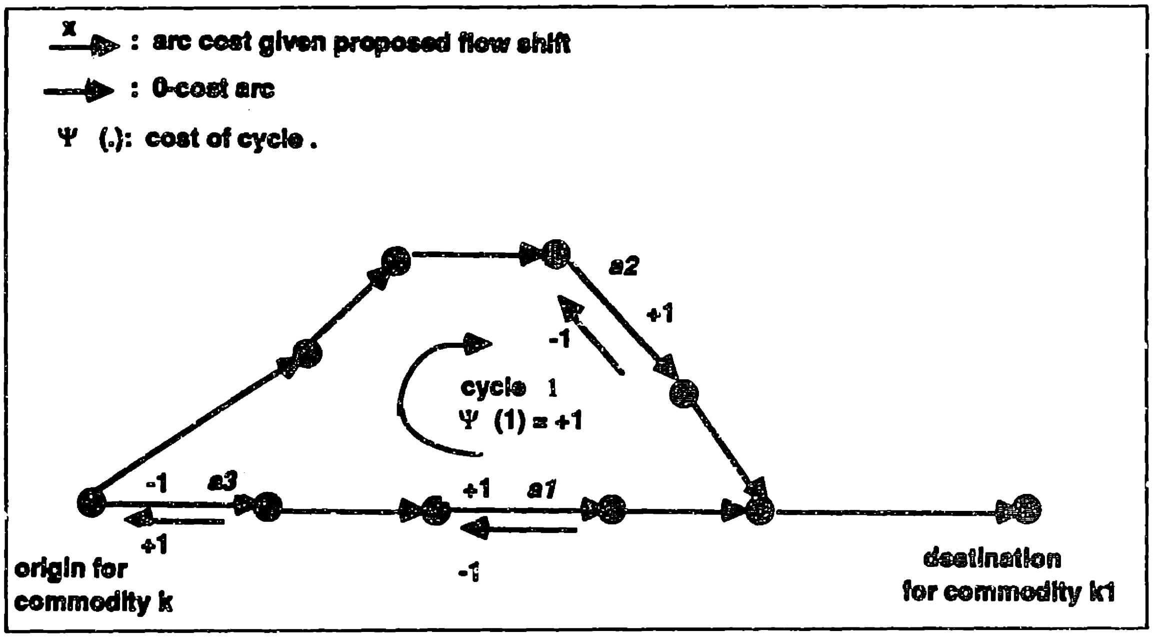

Figure 5.5a: Cycling in the Flow Adjustment Algorithm-- Part I

the over-capacitated arc a1 but over-capacitates arc a2 and under-capacitates arc a3 in the process. The value of ΔI associated with this flow shift is +1. In an attempt to make this value negative, the Flow Adjustment Algorithm selects cycle arc a2, adjusts its cost, locates cycle 2 and adds it to the composite cycle (Figure 5.5b).

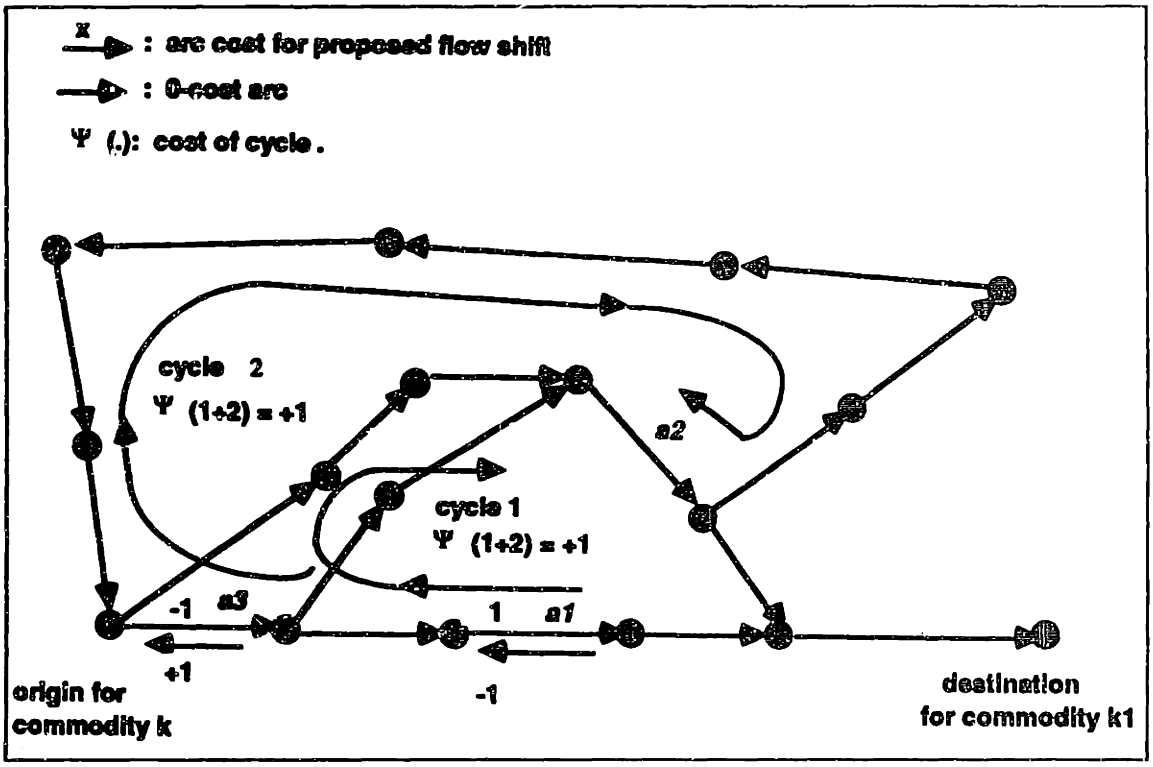

Figure 5.5b: Cycling in the Flow Adjustment Algorithm-- Part II

Cycle 2 removes the over-capacitating flow from arc a2 but further under-capacitates arc a3. As a result, for the composite shift consisting of cycles 1 and 2, ΔI remains equal to +1. Thus, the Flow Adjustment Algorithm continues by selecting arc a3, adjusting its cost and locating cycle 3 (the reverse of cycle 1), as shown in Figure

5.5c. Cycle shift 3 results in under-capacitating arc a2 and further over-capacitating arc a1. As a result, the value of ΔI is increased to +2. Consequently, the Flow Adjustment Algorithm proceeds by

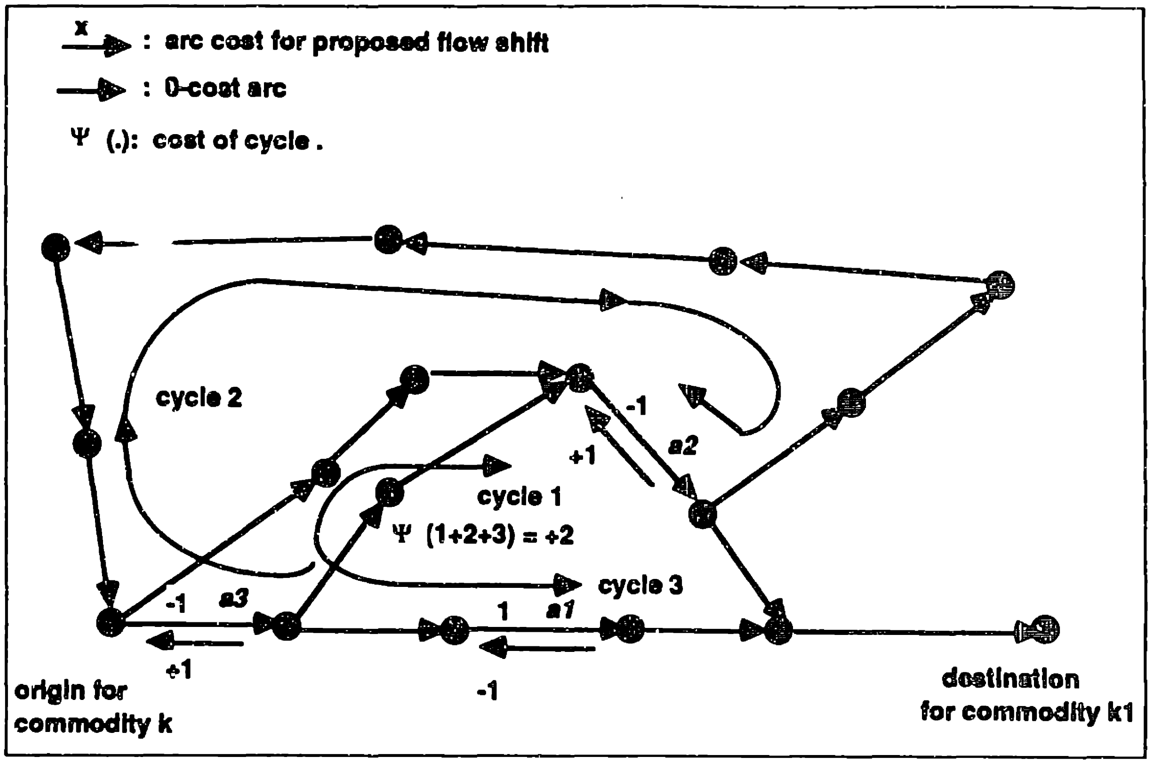

Figure 5.5c: Cycling in the Flow Adjustment Algorithm-- Part III

selecting arc a2, adjusting its cost to reflect its current status as an UCarc and locating cycle 4 (the reverse of cycle 2), as shown in Figure 5.5d. Notice that the composite cycle constructed to this point contains only cycles which counter the effects of one another--i.e. cycle 3 "undoes" the effects of cycle 1 and cycle 4 "undoes" the effects of cycle 2. As a result, a flow shift around the current composite cycle is equivalent to no flow shift at all. The Flow

Adjustment Algorithm process can continue, again selecting OCarc a1, and then repeatedly and endlessly finding cycles 1, 2, 3 and 4.

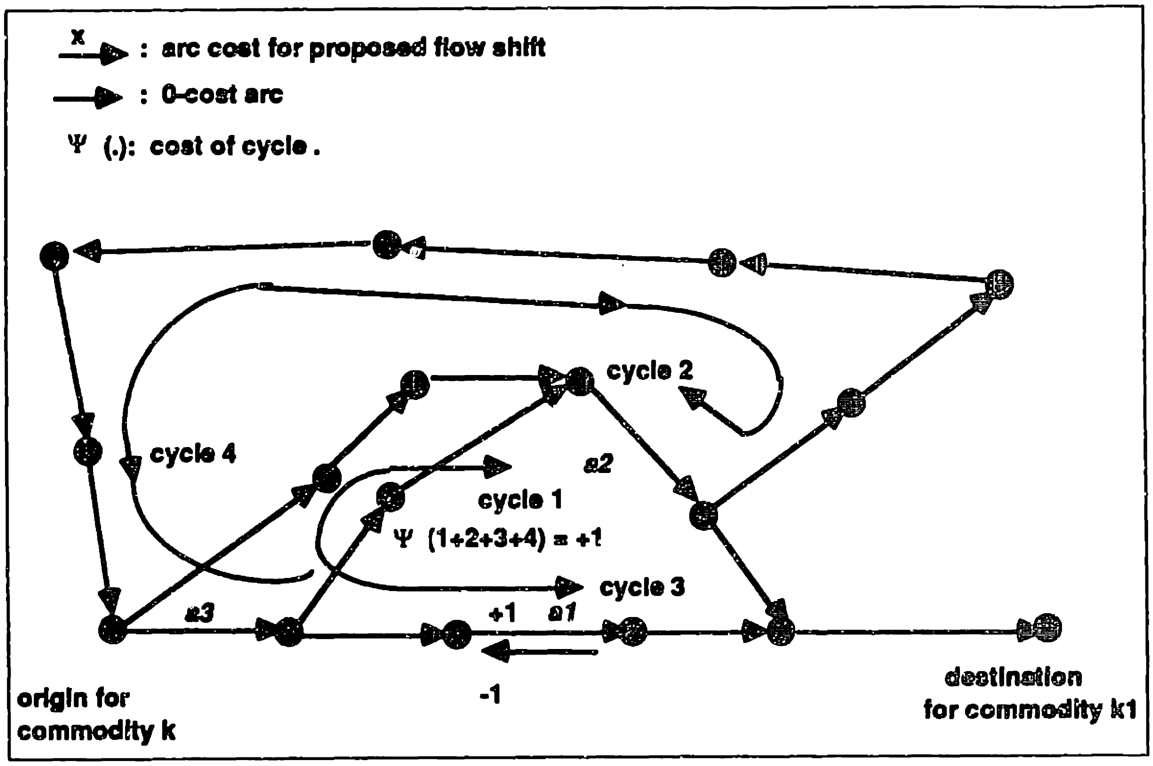

Figure 5.5d: Cycling in the Flow Adjustment Algorithm-- Part IV

To avoid cycling in the Flow Adjustment Algorithm, an upper limit is set on the number of cycle searches performed prior to the discovery of a negative cost composite cycle. If no flow shift is located, the search process begins from a different starting point by selecting a different initial problem arc, if one exists. If cycling continues to result, then, as before, to guarantee that only finitely many calls to the flow adjustment algorithm are necessary to solve the RP problem, the simplex method is invoked.

In summary, where insufficient progress is achieved using the flow adjustment algorithm, the RP/DRP problem is solved by applying the simplex method to the DRP problem (if problem size permits).

5.6 Summary

Each iteration of the Flow Adjustment Algorithm attempts to reduce the infeasibility measure by selecting a problem arc and searching for a flow shift which improves its condition while satisfying the algorithmic design features. If $\Delta I \geq 0$ for the located flow shift, then the condition of the selected problem arc is improved at the expense of at least one other arc with flow exactly at capacity (Theorem 5.1) This new problem arc is selected, its cost is modified, and a search to nullify the undesired results of the previous flow shift is launched. In a domino-like pattern, a succession of flow shifts is triggered, each flow shift undoing the damage caused by earlier shifts. If ΔI becomes strictly less than zero, a flow shift can be executed which strictly reduces infeasibility (Observation 5.2) and the algorithmic design criterion is achieved. If, however, ΔI remains positive and the algorithm terminates, then the accomplishment of the Flow Adjustment Algorithm is the elimination of all negative cost cycles in RN (Proposition 5.1) and the determination of an optimal solution to the RP/DRP problem (Theorem 5.4). In this situation, the algorithmic design criterion is satisfied, not in the Flow Adjustment Step, but rather, in the Price Adjustment Step (as demonstrated in Chapter 6).

If insufficient progress or cycling occurs in the search for a flow shift, the simplex algorithm can be invoked (if problem size permits) and the RP/DRP subproblems will be optimally solved. Otherwise, if the subproblems are too large to be solved using the simplex method, the PDN algorithm is not guaranteed to produce an optimal solution to the CMCF problem. Instead, the best the PDN algorithm can do is to provide a primal feasible solution to P, a dual feasible solution to D and thus, upper and lower bounds on the optimal solution value.

Chapter 6 presents the PDN algorithm and shows how the Flow Adjustment Algorithm fits into its framework.

6. The PDN Algorithm

Figure 6.1 depicts the PDN Algorithm. A "one-time" initialization step (Step 0) is followed by an iterative process wherein a restated problem, RP, is defined (Steps 1 and 4) and solved (Step 2). If, in solving the RP problem, the original problem is also solved, the algorithm terminates (Step 3). Otherwise, another RP problem is defined and solved. The process repeats in this manner until the original problem is optimally solved. The following sections (6.1 through 6.5) describe in detail the steps (shown in Figure 6.1) of the PDN Algorithm. Then, section 6.6 presents the PDN algorithm itself and Section 6.7 discusses finiteness. Finally, section 6.8 summarizes the PDN Algorithm.

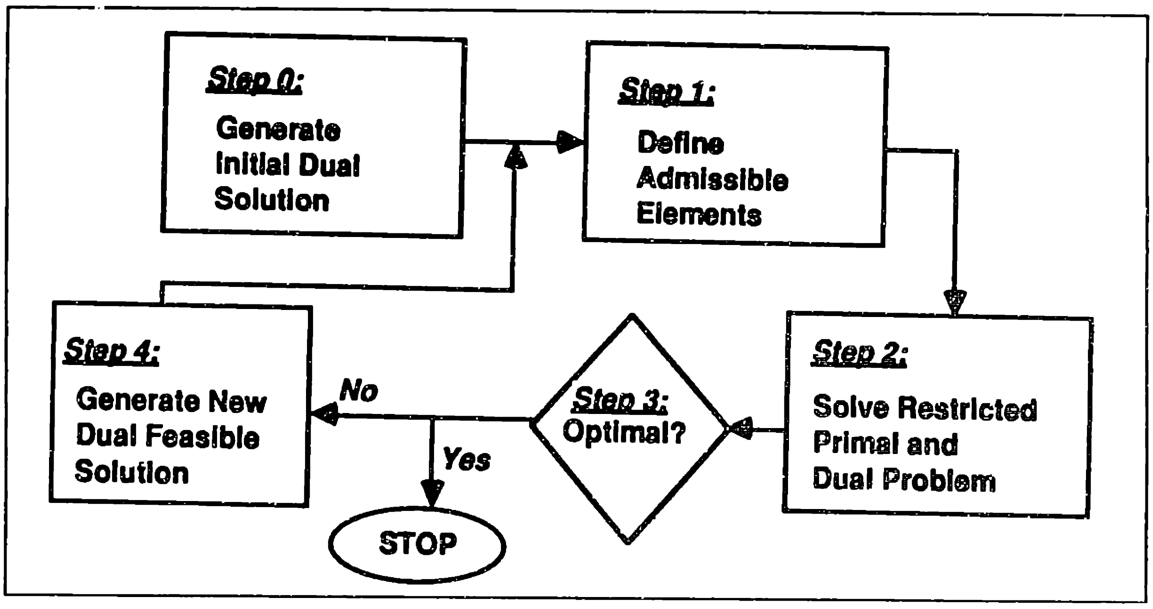
Step 0:
Generate Initial Dual Solution
Step 1:
Define Admissible Elements
Step 2:
Solve Restricted Primal and Dual Problem
Step 3:
Optimal?
No
Yes
Step 4:
Generate New Dual Feasible Solution
STOP

Figure 6.1: The PDN Methodology

6.1 PDN Step 0: Initialization

The purpose of the initialization component is to determine a set of initial dual prices and flow assignments which satisfy both dual feasibility (equations [2.6] through [2.8]) and complementary slackness conditions (equations [2.10] through [2.13]). These criteria can be easily satisfied. In fact, the only algorithmic tool necessary is a shortest path procedure. The initialization procedure is as follows:

Step 1) Set all dual prices to zero;

Step 2) Using these initial dual prices, run a shortest path algorithm for each commodity and assign all commodity flows to shortest paths without regard to arc capacity constraints.

Step 1 ensures the satisfaction of complementary slackness conditions #1 and #4 (equations [2.10] and [2.13] respectively). Dual feasibility and the satisfaction of demand requirements (equations [2.3]), flow non-negativity (equations [2.4]), and complementary slackness conditions #2 and #3 (equations [2.11] and [2.12]) are guaranteed by Step 2. In the initialization step, however, flow is assigned without regard to arc capacity constraints and thus, satisfaction of the bundle constraints (equations [2.2]) is not guaranteed.

To summarize, the initialization component provides initial dual and primal solutions which satisfy all of the design features of the PDN Algorithm.

6.2 PDN Step 1: Define Admissible Elements

The PDN algorithm determines the set J of admissible elements using the admissibility conditions defined for the primal-dual method (equations [2.14] and [2.15]). For a given dual feasible solution, admissible elements are:

i) shortest origin to destination paths for each commodity; and

ii) slack variables s_{ij} for arcs $ij \in A$ with dual arc price equal to zero.

These admissible elements are used to construct the **residual, restricted network, RN**, described in detail in Definition 4.4.

6.3 PDN Step 2: Solve RP and DRP Problem

Step 2 is motivated by Result #1 which states that any feasible solution to P exclusively utilizing the elements of the admissible set J, is an optimal solution to P. Thus, the objective of the RP problem is to find a flow assignment in the residual, restricted network which minimizes total infeasibility. Chapter 5 showed that repeated executions of the primal-based Flow Adjustment Step can achieve this objective. Each Flow Adjustment Step terminating with a flow shift, strictly advances the primal solution closer to feasibility (Observation 5.2) and therefore, to optimality in P. A Flow Adjustment Step failing to locate an infeasibility reducing flow shift and instead, terminating with the elimination of all negative-cost cycles in RN, optimally solves the RP/DRP problem. If repeated executions of the Flow Adjustment Algorithm reduce the infeasibility measure to zero or eliminate all negative-cost cycles in RN, the RP/DRP problems are optimally solved and the iterative Flow Adjustment Step terminates. If cycling or insufficient progress results in the solution process, the PDN algorithm either invokes the simplex algorithm to solve the RP/DRP problem or terminates prematurely with a non-optimal solution.

6.4 PDN Step 3: Test for Optimality

Step 3 performs the test for optimality by computing the value of infeasibility associated with the flow assignment generated in Step 2. By Corollary 4.1, the CMCF problem P is optimally solved if the infeasibility measure I is equal to zero. Thus, the PDN Algorithm terminates if $I = 0$, otherwise $I > 0$ and feasibility in P cannot be achieved for the current solution for D.

6.5 PDN Step 4: Generate New Solution for D

If feasibility in P cannot be achieved for the current dual solution, a new dual solution for D is generated using a dual-based Price Adjustment Step. The purpose of the Price Adjustment Step is to move the dual solution closer to optimality in D and to create the possibility for an improved (i.e. less infeasible) primal solution for P. These criteria are satisfied by defining the PRICE ADJUSTMENT STEP as follows:

Definition 6.1: The Price Adjustment Step adjusts the dual prices, π, on a set of arcs in RN such that:

i) the dual arc prices, π^a, remain non-negative;

ii) all utilized admissible elements remain admissible;

iii) at least one inadmissible element becomes admissible; and

iv) the dual objective function value increases.

To satisfy the requirement that all utilized admissible elements remain admissible, the Price Adjustment Step must ensure that all flow carrying shortest paths remain shortest; and must not increase dual prices on arcs with total flow less than capacity.

To satisfy the requirement that at least one inadmissible element become admissible, the Price Adjustment step must either ensure that at least one longer path becomes a shortest path for some commodity or at least one dual arc price is reduced to zero.

The requirement that the price adjustment step augment the set of admissible elements is significant because by enhancing RN, it creates the possibility for a flow shift which can lead to a reduction in primal infeasibility.

6.5.1 Satisfaction of Design Features and Criterion

The following points show that the price adjustment step of Definition 6.1 satisfies all of the design features of the PDN algorithm:

Point 1) **Dual feasibility** (equations [2.6] through [2.8]) is always satisfied because by definition, arc prices are restricted to non-negative values and the commodity-specific dual nodes prices are correctly determined using a shortest path procedure;

Point 2) Since all of the utilized admissible elements remain admissible, **complementary slackness conditions #3** (equations [2.12]) are satisfied for the current flow assignments;

Point 3) Since the price adjustment step does not alter any of the primal flow assignments, demand (equations [2.3]) and flow non-negativity (equations [2.4]) requirements are satisfied.

The algorithmic design criterion requires advancement of the solution towards optimality at each iteration of the PDN Algorithm. By Definition 6.1, the price adjustment step is required to increase the dual objective function value, thus advancing it towards optimality.

In addition to improving the dual solution, the Price Adjustment Step is instrumental in the improvement of the primal solution. It alters the set of shortest paths and admissible slack variables so as to create the potential for a Flow Adjustment Step to reduce the infeasibility measure.

The next section describes the implementation of the Price Adjustment Step and shows that the requirements of definition 6.1 are satisfied.

6.5.2 Implementation

A dual price adjustment step is invoked (as shown in Figure 6.1), when a Flow Adjustment Step fails to locate an infeasibility-reducing flow shift. As shown in Chapter 5, using the arc costs generated in the Failed Flow Adjustment Step, an optimal solution to the DRP problem can be easily constructed. Then, as in the Primal-Dual algorithm presented in Chapter 2, the optimal prices for the DRP problem are used in the Price Adjustment Step to create a new solution for D. The new solution is achieved by adjusting the current solution using equations [2.32] as follows:

$$\pi'_{ij} = \pi_{ij} + \theta_i \Delta\pi_{ij}, \quad ij\in A; \qquad \text{and}$$
$$\sigma^{k'} = \sigma^k + \theta_i \Delta\sigma^k, \quad k\in K.$$

where:

π_{ij}, σ^k: current dual prices for D;

π'_{ij}, $\sigma^{k'}$: revised dual prices for D;

$\Delta\pi_{ij}$, $\Delta\sigma^k$: optimal solution to the DRP problem-- outputs of the failed Flow Adjustment Step;

θ_i: multiplier at iteration i.

The value of the multiplier θ_i is determined using equation [2.33]:

$$\theta_i = \text{MIN}\ [(\text{MIN}_\alpha(c_p{}^k + \Sigma_{ij\in A}\ \pi_{ij}\delta_{ij}{}^{p,k} - \sigma^k)/$$
$$(-\Sigma_{ij\in A}\ \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k));\ \text{MIN}_\beta(-\pi_{ij}/\Delta\pi_{ij})]$$

where:

$\alpha = \{p|p\in(P^k),\ \forall k\in K,\ -\Sigma_{ij\in A}\ \Delta\pi_{ij}\delta_{ij}{}^{p,k} + \Delta\sigma^k = \gamma_p{}^k > 0\};$

$\beta = \{ij|ij\in A \cap \Delta\pi_{ij}<0\};$

For the set β, the value of θ_i corresponding to each arc $ij\in\beta$ is determined simply by computing the ratio of π and $\Delta\pi$. In the case of the set α, however, the value of θ_i must be computed for each path p and commodity k with $\gamma_p{}^k > 0$. Since there are potentially an exponential number of such computations, a label-correcting algorithm was developed. This algorithm, by searching the arcs of RN (and not the paths) determines and sets node prices which are used to expedite the calculation of the minimum θ_i for the set of paths $p \in \alpha$.

Using the same arguments presented in Result #3, the revised dual prices for D, π' and σ', satisfy all feasibility requirements (equations [2.6] through [2.8]) and maintain admissibility for all basic elements in the solution to the ith RP problem. Similarly, using the same arguments presented in the proof of Result #4, at least one inadmissible element becomes admissible with the revised dual prices. Finally, the arguments presented in the proof of Result #5 can be used to show that, barring degeneracy, the revised dual prices achieve ascent of the objective function of the CMCF dual problem D. This statement can be unqualified as follows:

<u>Theorem 6.1:</u> In the PDN Algorithm, each Price Adjustment Step leads to strict ascent of the objective function value for D.

<u>Proof:</u> From Result #5, if dual prices are altered by a non-zero amount (equations [2.32] and [2.33]), ascent of the objective function value for D is achieved. Dual prices are unaltered by the Price Adjustment Step only if $\Delta\pi$ equals zero for each arc or θ_i equals zero. First, it is impossible for $\Delta\pi$ to equal zero for each arc because the Price Adjustment Step is invoked only if optimality is not achieved and the Flow Adjustment Step fails to locate an infeasibility reducing flow shift. Since infeasibilities exist for the current flow assignment, there is at least one OCarc or UCarc. Thus, by design of the

Flow Adjustment Algorithm and equations [4.31], at least one arc has a non-zero $\Delta\pi$ value. Second, it is impossible for the value of θ_i to equal zero. This can be seen be examining equation [2.33] which shows that θ_i is either equal to:

i) the difference between the shortest path $p \notin J(k)$ and the shortest path $p \in J(k)$ divided by a non-zero quantity; or

ii) the ratio of two non-zero quantities.

In the PDN algorithm, the admissible set J includes <u>all</u> shortest paths. Thus, the difference between the shortest path not in set J and the shortest path in set J is a non-zero quantity. Hence, θ_i is a non-zero quantity. It follows that at least one dual price is altered by a non-zero amount in the Price Adjustment Step of the PDN algorithm and hence, by Result #5, strict ascent of the dual objective function for D is attained. ■

Hence, Result #3 and Theorem 6.1 and Results #4 and #5, show that a price adjustment using equations [2.32] and [2.33], together with the current dual solution and information generated in the failed flow adjustment step, meets the specifications of Definition 6.1, thereby satisfying the algorithmic design features and criterion.

<u>6.6 Algorithm</u>

In summary, the PDN algorithm begins with a feasible solution to the dual CMCF problem, D. This dual feasible solution defines the set of admissible elements used to build the restricted, residual network, RN, over which all flow is assigned. Flow Adjustment Steps shift flow among the paths in RN such that conservation of flow is maintained and total primal infeasibility is reduced. These Flow Adjustment Steps are repeatedly performed until the infeasibility measure is reduced to zero or until no further reduction in infeasibility is possible. (If insufficient progress is achieved, it may be necessary to use the simplex algorithm to ensure that one of these two conditions is achieved.) If infeasibility is eliminated (i.e. I = 0), the CMCF problem is optimally solved and the PDN Algorithm terminates. Otherwise, the dual feasible solution for D is altered in the Price Adjustment Step and the entire process repeats. The pseudo-code for the PDN Algorithm is presented in Figure 6.2.

6.7 Finiteness

Theorem 6.2: The PDN Algorithm (using standard perturbation techniques in the simplex method to bar degeneracy) optimally solves the CMCF problem in a finite number of steps.

Proof: Let an iteration of the PDN Algorithm be defined by the outer DO WHILE loop shown in Figure 6.2. Then, each algorithmic iteration includes a series of Flow Adjustment Steps and a Price Adjustment Step. Assuming that the simplex algorithm can be used to solve the RP or DRP subproblems if necessary and using Theorem 5.4, the primal and dual solutions existing at the termination of the failed Flow Adjustment Step optimally solve the RP and DRP problems, respectively. Results #6 and #7 show that the optimal solution to the RP problem, when the simplex method is used, is a basic feasible solution to P.

```
Program PDN Algorithm

Given:  $\pi$, $\sigma$ dual feasible

infeas ← .false.
opt ← .false.

DO WHILE (.not.opt .and. .not. infeas)  !Until optimal
     solved ← .false.

     CALL DEFINE_SET_J                  !Finds admissible paths

     DO WHILE (.not.solved)             !Until solve RP/DRP
          solved ← .false.
          fail   ← .false.

          CALL Flow_Adjust(fail)        !Shift flows

          IF (fail) THEN
             Call Simplex               !Solve RP/DRP Problem
             solved ←  true.            !with Simplex Method
          ELSE IF (I = 0 .or. Stack S = Ø) THEN
             solved ←  true.            !Solved RP/DRP w/ FAA
          END IF
     END DO

     IF (I = 0) THEN                    !Optimality test
         opt ← .true.
     ELSE IF ($\gamma^k \le 0\ \forall p \in J^k$, $\forall k \in K$ .and. $\Delta\pi_{ij} \ge 0\ \forall ij \in A$) THEN
         infeas ← .true.
     ELSE
         CALL CALC_$\theta_i$                 !Calculate optimal  value
         Call Price_Adjust              !Update dual prices
     END IF
 END DO

 RETURN
 END.
```

Figure 6.2: Pseudo-code for the PDN Algorithm

Although the PDN algorithm is not guaranteed to generate basic solutions, the arguments used in the proof of Results #6 and #7 can be used to show that the optimal solution to the RP problem generated by the PDN Algorithm can be considered a solution to the CMCF problem, P. Thus, it can be observed that for each solution to P generated in the PDN algorithm, there exists a basic feasible solution to P with equal z^*_{RP} value. Applying the arguments used in the proof of Result #6 and using perturbation techniques to eliminate degeneracy in the simplex method, the optimal objective function value obtained by the PDN Algorithm for the i+1st RP problem is strictly less than the corresponding value for the ith RP problem. Thus, the failed Flow Adjustment Step generates solutions at each iteration which have decreasing z^*_{RP} values and are therefore, non-repeating. Corresponding to each of these solutions is a basic feasible solution for P with equal z^*_{RP} values. Thus, for any sequence of solutions to P generated by the PDN algorithm, there exists a corresponding sequence of non-repeating basic feasible solutions to P. Hence, the PDN problem optimally solves the CMCF problem in a finite number of steps. ∎

At each iteration of the PDN algorithm, the Flow Adjustment Algorithm must be repeatedly executed to solve the RP/DRP problem. To expedite this solution process, advance start solutions are employed by the Flow Adjustment Algorithm. The advance start is achieved by taking the solution existing at the termination of the preceding Flow Adjustment Algorithm as the initial solution for the current Flow

Adjustment Algorithm. For the current admissible set, this initial solution satisfies all the algorithmic design criteria irrespective of the manner in which the preceding Flow Adjustment Algorithm terminated (Result #6). If the Flow Adjustment Algorithm terminated with a flow shift, the set of admissible elements is unchanged and the flow assignment continues to satisfy all of the design features. Similarly, if the preceding Flow Adjustment Algorithm terminated with failure to locate a flow shift, the set of admissible elements is altered by a follow-on Price Adjustment Step but all utilized admissible elements remain admissible and hence, all flow remains assigned to shortest paths. Thus, the solution existing at the termination of the Flow Adjustment Step is a valid starting solution for the succeeding Flow Adjustment Step.

6.8 Summary

The need to solve large-scale CMCF problems of the size encountered in practice motivated the development of a new network-based algorithm named PDN. The PDN algorithm is modeled after the primal-dual method and therefore, never directly solves the original problem. Instead, it indirectly solves the original problem by repeatedly solving smaller, simpler restricted problems.

The algorithm begins by using a given dual feasible solution in D to define admissible elements. Then it tries to construct, through flow adjustment steps, a feasible primal solution for P using only these admissible elements. Flow adjustment steps are repeatedly executed, each one moving the primal solution quantifiably closer to feasibility, until no further improvement in the primal solution can be achieved and the RP problem is optimally solved. To guarantee that the RP problem is solved to optimality, the PDN algorithm relies upon the assumption that the simplex method is capable of solving problems of the DRP (or RP) problem size. If this assumption does not hold, the PDN algorithm is not guaranteed to optimally solve the CMCF problem. Instead, the PDN algorithm constructs a primal feasible solution to P and determines both upper and lower bounds on the optimal solution value.

If the restricted problem size is not prohibitive, the PDN algorithm is **guaranteed** to optimally solve the RP/DRP problems at each iteration. (For very large problems which cannot be solved using the simplex method, this guarantee does not hold because the Flow Adjustment Algorithm may get "stuck". If this situation does not occur, the PDN algorithm will optimally solve the RP/DRP problems at each iteration.) If the optimal primal solution to the RP problem is feasible for the original CMCF problem, the PDN algorithm terminates with optimality in P. If not, the dual prices are adjusted and the entire process repeats.

7. Experimental Design

Several solution methods for the capacitated, multi-commodity network flow problem were investigated. These solution methods are classified within the following major categories:

i) LP methodology;

ii) DW Methodology;

iii) PD Methodology; and

iv) PDN methodology.

For each methodology, different algorithmic strategies were implemented and then used to solve a variety of CMCF test problems. The test problems differ in network topology; number of commodities; and amount of congestion in the network. Several performance measures are used to evaluate the performance of each algorithmic implementation.

Sections 7.1 through 7.5 describe the characteristics of the CMCF algorithmic implementations and test problems. Section 7.6 then defines the computing environment. Finally, Section 7.7 describes the performance measures used to evaluate and compare the algorithms.

7.1 LP Solution Methodology

The LP Solution Methodology formulates the CMCF problem using the node-arc formulation described in equations [1.1] - [1.4]. The resulting linear program (LP) is then directly solved without any form of decomposition using MINOS: "A Modular In-Core Nonlinear Optimization System" [41]. (MINOS implements, among other things, the simplex method.)

Of all four solution methodologies tested, the LP solution method is the most general. It does not use specialized decomposition strategies and does not exploit the specific structure of the CMCF problem. The purpose of the LP solution method is to provide a "base" to which other solution methods can be compared.

7.2 DW Solution Methodology

The DW solution methodology solves the CMCF problems using the Dantzig-Wolfe algorithm (Chapter 1). The Dantzig-Wolfe Algorithm was implemented because Assad [5,6,7] and Swoveland [49,50] reported that price-directive decomposition (such as Dantzig-Wolfe) outperforms primal partitioning and resource-directive solution methodologies.

The DW methodology involves LP subproblems which are solved using the XMP [39] software package. XMP is particularly well-suited for the column-generation strategy used in Dantzig-Wolfe decomposition. When the solution process is interrupted by the addition of columns, XMP incorporates these columns into the simplex tableau and resumes pivoting, beginning with the solution existing prior to the addition of the columns.

At each iteration of the DW method, the added columns are generated using Moore's [40] label-correcting shortest path algorithm with Pape's [43] suggested sequence list management scheme. This shortest path algorithm is performed on a network which is generated from the input data and stored in forward star form (Bradley, et. al. [12]).

7.3 PD Solution Methodology

The PD Solution Methodology solves CMCF problems using the primal-dual methodology, described in Chapter 2. As in the DW solution method, **XMP** is used to solve the restricted subproblems and Moore-Pape's label-correcting algorithm is used to generate the new columns at each iteration.

Two distinct primal-dual schemes, using different RP problem formulations, were implemented and tested. The objective was to assess the impact of the RP problem formulation on efficiency of the primal-dual method.

In the first implementation, called PD1, the RP problem is formulated with equations [2.16] - [2.20]. Two artificial variables, $w^{+}_{\cdot}$ and $w^{-}_{\cdot}$, are added for each bundle constraint in P (equations [2.12]). The artificial variables associated with the bundle constraint for each arc ij can be interpreted as parallel, opposing **artificial** arcs which permit additional flow to travel from node i to node j or from node j to node i. These artificial arcs allow flow to over-capacitate or under-capacitate the original arcs and thus, create the possibility for primal infeasibilities in the form of OCarcs and UCarcs. The objective is to eliminate all of the infeasibilities by reducing the flow on each artificial arc to zero.

In the second primal-dual implementation, called PD2, the RP problem is formulated by adding a single artificial variable, denoted $w^{+}_{\cdot}$, to each equation in P, i.e.:

$$\text{MIN } z_{RP} = \Sigma_{ij\in A}\, w_{ij}^{+} + \Sigma_{k\in K}\, w_k^{+} \qquad [7.1]$$

subject to:

$$\Sigma_{k\in K}\, \Sigma_{p\in J^k}\, x_p^k \delta_{ij}^{p,k} + w_{ij}^{+} = d_{ij}, \quad \forall\ ij\in IJ^{+}(\pi); \qquad [7.2a]$$

$$\Sigma_{k\in K}\, \Sigma_{p\in J^k}\, x_p^k \delta_{ij}^{p,k} + s_{ij} + w_{ij}^{+} = d_{ij}, \quad \forall\ ij\in IJ^{+}(\pi)^c; \qquad [7.2b]$$

$$\Sigma_{p\in J^k}\, x_p^k + w_k^{+} = b^k, \quad \forall\ k\in K; \qquad [7.3]$$

$$x_p^k \geq 0,\ \forall\ p\in J^k,\ \forall\ k\in K;\ x_p^k = 0,\ \forall\ p\in (J^k)^c,\ \forall\ k\in K; \qquad [7.4]$$

$$w_{ij}^{+} \geq 0,\ \forall\ ij\in A;\ \ w_k^{+} \geq 0,\ \forall\ k\in K. \qquad [7.5a]$$

$$s_{ij} \geq 0,\ \forall\ ij\in A;\ \ s_{ij} = 0,\ \forall\ ij\in IJ^{+}(\pi). \qquad [7.5b]$$

where:

$w_{.}^{+}$: artificial variables measuring infeasibility of the flow assignment.

The artificial variables in the bundle constraints of the PD2 formulation (equations [7.2]) can be interpreted again, as artificial arcs. This time, however, since only one artificial variable exists per constraint, the artificial arcs can create the possibility for infeasibilities in the network in the form of UCarcs only, and not OCarcs. If insufficient shortest path capacity exists for some commodity k, the PD2 formulation, rather than permit the over-capacitation of an arc in RN, assigns the excess flow to the artificial variable w_k^{+}. Since w_k^{+} is the artificial variable associated with the demand constraint for commodity k (equations [2.3]), w_k^{+} can be interpreted as an artificial arc spanning from the origin to the destination of commodity k. This artificial origin to destination arc holds the flow of commodity k which cannot be feasibly assigned to the restricted network. (Hence, artificial arcs containing flow can be considered OCarcs since the "true" capacity of these arcs is zero.)

Parallels can be "loosely" drawn between the DW solution methodology and the PD2 algorithmic implementation using the RP formulation of equations [7.1] - [7.5]. In both cases, commodity flow which cannot be feasibly assigned to the restricted network is instead assigned to the artificial origin-to-destination arcs. These artificial arcs have essentially infinite cost and thus, the objective is to find a flow assignment not utilizing any of these artificial arcs. The parallels between the two methods cannot, however, be taken too far. Although both methods formulate the problem similarly, the way in which the two methods define the restricted network and achieve the common objective is significantly different.

7.4 PDN Solution Methodology

The PDN Solution Methodology solves the CMCF problem using the Primal-Dual Network algorithm, described in Chapters 3 through 6.

In the interest of increased efficiency, a commodity is defined, in all PDN implementations, as a set of demands originating at a common node. A commodity then, is represented by a tree of origin to destination demands (all rooted at a common origin). Efficiencies are gained using this tree representation because the number of required executions of such routines as the shortest path is reduced.

Both the PDN and LP solution methodologies can effortlessly accommodate the tree representation of commodities. The PDN implementations are completely network based and are thus, unaffected by a commodity having several destinations rather than a single destination. Similarly, the LP solution method formulates the CMCF problem in node-arc form and can thus, easily accommodate either the path or the tree commodity definition. The DW and PD implementations, however, use the path formulation of the CMCF problem and employ column generation techniques where each column is associated with a specific path for one origin to destination demand. The value of the LP variable corresponding to any particular column (or path) measures the amount of total flow on that path. Hence, in the DW and PD solution methodologies, commodities are represented as origin-to-destination pairs and not origin or destination-based trees.

For the PDN solution methodology, like the PD solution methodology, the effect of the RP problem formulation on algorithmic performance was tested. The PDN algorithm using the RP formulation of equations [2.16] - [2.20] was implemented and is referred to as PDN1. PDN2 is a variation of PDN1 which uses the RP formulation of equations [7.1] - [7.5]. Thus, the PDN1 implementation, like PD1, permits the existence of both OCarcs and UCarcs in RN. PDN2, like PD2, allows arcs in RN to be UCarcs but permits only the artificial origin to destination arcs to be OCarcs. PD1 and PDN1, like PD2 and PDN2, have the same problem formulation and solution strategy. The only difference between the corresponding PD and PDN formulations is in the solution of the RP subproblems: the PD methodology uses the simplex algorithm and the PDN methodology uses repetitions of the Flow Adjustment Algorithm (Chapter 5).

For the PDN1 and PDN2 implementations, the effect of flow shift strategies on algorithmic efficiency was also tested. PDN1a and PDN1b (similarly PDN2a and PDN2b) refer to two distinct implementations of PDN1 (PDN2) which differ in the flow shift criterion exercised. In PDN1a (PDN2a), each flow adjustment step is required to strictly reduce overall infeasibility (i.e. $\Delta z_{RP} = \Delta I < 0$) without increasing infeasibility on any arc (i.e. $\Delta I_{ij} \leq 0$, $\forall ij \in A$; where ΔI_{ij} is the change in the infeasibility measure for arc ij resulting from the flow shift.) Thus, in PDN1a (PDN2a), each flow shift reduces the flow on an OCarc or increases the flow on an UCarc without (further) over-capacitating or (further) under-capacitating any other arc in RN. In PDN1b (PDN2b), as in PDN1a (PDN1b), each flow adjustment step is required to strictly reduce total infeasibility in RN. In PDN1b (PDN2b), however, no additional restrictions are placed on individual arc infeasibilities. As long as total infeasibility is reduced (and adherence to the RP formulation guidelines is maintained), the infeasibility level for a specific arc can improve, worsen or stay the same. Thus, PDN1b (PDN2b) is, in a sense, a relaxation of PDN1a (PDN2a).

Table 7.1 summarizes the characteristics of the PDN algorithmic implementations.

	PDN1		PDN2	
	a	b	a	b
Permitted OCarcs in RN?	Yes	Yes	No	No
Artificial OD arcs for OCarcs?	No	No	Yes	Yes
$\Delta z_{RP}<0$ and $\Delta I_{ij}\leq 0$ $\forall ij \in A$?	Yes	No	Yes	No
$\Delta z_{RP}<0$ only?	No	Yes	No	Yes

Table 7.1: Variations on the PDN Algorithmic Implementations

7.5 Test Data

Each of the algorithmic implementations described in Sections 7.1 through 7.4 was run on a set of test problems. The test problems, called P1, P2, P3 and P4, were constructed from the set of problems used by Assad [5,7]. Their general characteristics are shown in Table 7.2. Notice that the problem sizes are sufficiently small so as to allow their solution with the simplex method.

Problems P1 through P4 differ in network topology and the ratio of capacitated arcs to total number of arcs. (The structure of the networks parallel those used by Swoveland [49,50].) Problems P1 and P3 have essentially acyclic network structure and thus, model applications to multi-commodity, multi-period, production-distribution problems or to problems involving the scheduling of vehicles on a time-space network. Problems P2 and P4 have the structure of undirected communications networks.

Problem	# Nodes	# Arcs	# Capacitated Arcs
P1	47	98	98
P2	25	96	96
P3	83	205	95
P4	44	268	98
REAL	1195	2820	1256

Table 7.2: Characteristics of the Test Problems

In problems P1 and P2 all arcs are capacitated whereas in problems P3 and P4, the ratio of capacitated arcs to total number of arcs is approximately 0.48 and 0.38 respectively.

For each network topology (i.e. problem classification), there is an associated set of test problems in which the number and the size (demand) of commodities are varied. Table 7.3 shows the variations in commodity data for problems P1 through P4.

To evaluate the performance of the various algorithmic implementations and the effect of network topology and commodity characteristics, all of the algorithmic implementations were run on every network for each variation in commodity data.

Problem	# of Commodities		Total Demand
	paths	trees	
P1.1	3	3	28
P1.2	5	3	28
P1.3	7	3	28
P1.4	6	3	30
P1.5	10	3	30
P1.6	15	3	30
P2.1	10	8	60
P2.2	14	12	84
P2.3	20	15	120
P3.1	6	6	36
P3.2	12	6	72
P3.3	6	6	105
P4.1	6	6	90
P4.2	10	8	150
P4.3	14	11	210
P4.4	20	14	300
P4.5	30	17	450
REAL	2227	50	9751347

Table 7.3: Variations within Test Problems

Furthermore, since the PDN algorithmic development was motivated by the desire to solve "real-world" size problems, another test problem, called REAL, was generated using data provided by a large U.S. trucking firm. The data, which corresponds to a freight assignment problem, includes information concerning the required pick-up and delivery of less-than-truckload shipments. In servicing these shipments, the trucking firm incurs a per-mile charge. Thus, to minimize the number of miles driven, the objective is to achieve maximum in-vehicle consolidation (subject, of course, to level of service constraints). The potential for consolidation is enhanced by the company's use of consolidation terminals. These consolidation terminals, located throughout the country, perform the function of gathering together shipments destined for the same location.

The trucking company uses forecasted demands to define and fix routes for each vehicle. Once the vehicle routes are fixed, the transportation costs are considered "sunk" and the objective is to minimize the total service time for the shipments. This problem can be formulated as a CMCF problem where the vehicle routes, capacities and travel times define the network over which the shipments (commodities) are to be serviced and the objective is to minimize total time (cost). As shown in Tables 7.2 and 7.3, the company's multi-commodity network flow problem contains 1195 nodes, 2820 arcs (of which 1256 have finite capacity limitations) and 2227 demands (shipments).

7.6 Computing Environment

The computational testing was performed on a Digital VAXStation II operating under the MicroVMS V4.7 operating system. All of the algorithmic implementations were programmed in Vax Fortran V4.5 and all run times were determined (with no users on the system) using SECNDS--the intrinsic subprogram which calculates clock time.

7.7 Performance Measures

To evaluate the performance of the various algorithmic implementations, total run time and total number of iterations were recorded. In the evaluation process, each test problem was solved to optimality using each of the algorithmic implementations. (The PDN implementations optimally solved all of the test problems without resorting to the simplex algorithm.)

7.7.1 Number of Iterations

Caution must be exercised in comparing, among algorithmic implementations, the statistics reporting total number of iterations. An iteration is not defined the same for all of the alternative solution methods. For example, in the LP solution implementation, the number of iterations corresponds to the number of pivots of the simplex method. With respect to the remaining solution methods, however, the number of iterations corresponds to the number of times a new RP problem is generated. Even with this "common" definition, there is a difference between an iteration of the PDN method and an iteration of the DW and PD methods. In the DW and PD solution methodologies, an iteration corresponds to the solution of a RP subproblem and the follow-on dual price adjustment step. In the PDN solution methodology however, an iteration corresponds to finding and executing a dual price adjustment step which ascends the dual objective function. Hence, at the termination of a PDN iteration, the RP problem is not necessarily solved. In fact, it may be that only one commodity was involved in the iteration. In the extreme case, to solve the RP problem, the PDN algorithm may require one iteration for each commodity. In the PD and DW solution methodologies, however, at most one iteration is required to solve the RP problem.

To summarize, given the definitions of an iteration and holding all else constant, the total number of iterations is expected to be greater for the PDN implementations than for either the DW or PD implementations.

7.7.2 Run Time

Run time is another measure used in comparing the algorithmic implementations. Run time is the total amount of time required to read the input data and solve the resulting problem. Run time can be divided into three mutually exclusive time components. These time components, whose sum equals total run time, are called:

i) Initialization;

ii) RP Solution; and

iii) Dual.

The initialization time component measures the amount of time required to read in the input data and to manipulate the data into the required form necessary for the algorithmic implementation. In the DW, PD and PDN methodologies, the data manipulation involves generation of the network over which commodities flow. In addition, for the simplex-based methods, i.e. LP, DW and PD, the data manipulation involves the generation of the original simplex tableau.

The RP solution time component, (relevant for all but the LP solution methodology), measures the total amount of time expended in solving the RP subproblems. This value measures the amount of time the simplex algorithm is run in the DW and PD methodologies and the amount of time the Flow Adjustment algorithm is run in the PDN methodologies.

Finally, the dual time component (also relevant for all but the LP solution methodology) measures the amount of time necessary to either prove optimality or find new paths (i.e. generate new columns). For the DW implementation, this dual adjustment component involves only the execution of a shortest path algorithm for the commodities. In the PD and PDN implementations, however, this component involves the determination of θ (equation 2.33); the adjustment of dual prices; and then finally, the execution of a shortest path algorithm for the commodities.

To summarize, total run time is the sum of the times required for initialization, solution of RP subproblems, and adjustment of dual prices followed by generation of new paths (columns).

The next chapter describes the results of the computational experiments.

8. Computational Results

The algorithmic implementations presented in Chapter 7 were run on the set of test problems described in Tables 7.2 and 7.3. In this chapter, the results of these computational experiments are reported and analyzed. Section 8.1 analyzes the relationship between run time and the number of iterations. Then, Sections 8.2 through 8.4 examine the effects of variations in the problem data on algorithmic performance. Specifically, the results of the PDN1a algorithmic implementation are analyzed to determine the impact of total demand, number of demands, and average demand size on total run time. (PDN1a, as explained in Chapter 7, permits the existence of both OCarcs and UCarcs and performs flow shifts which strictly reduce total infeasibility without increasing the infeasibility for any one arc.) The PDN1a results are examined in particular because:

i) the PDN algorithm is newly developed; and

ii) the performance of the PDN1a implementation (as shown later in this chapter) is representative of the PDN implementations.

The effects of variations in the flow shift criterion are examined in Section 8.5. Then, in Sections 8.6 through 8.10, the performance of the four methodologies, (i.e. LP, PDN, PD and DW) are evaluated and compared. Next, Section 8.11 reports results concerning the solution of the large scale problem (Real) introduced in Chapter 7. Finally, Section 8.12 summarizes the conclusions drawn from the computational experiments.

8.1 Run Time vs. Number of Iterations

Intuitively, one would expect that run times are heavily influenced by the number of iterations performed to achieve optimality. As the number of iterations increases, run times should correspondingly increase and likewise, as the number of iterations decreases, run times should decrease. This reasoning is substantiated by Figures 8.1 and 8.2. The number of iterations and the run time plots follow the same general pattern. This indicates the strong correlation between run time and the number of iterations performed.

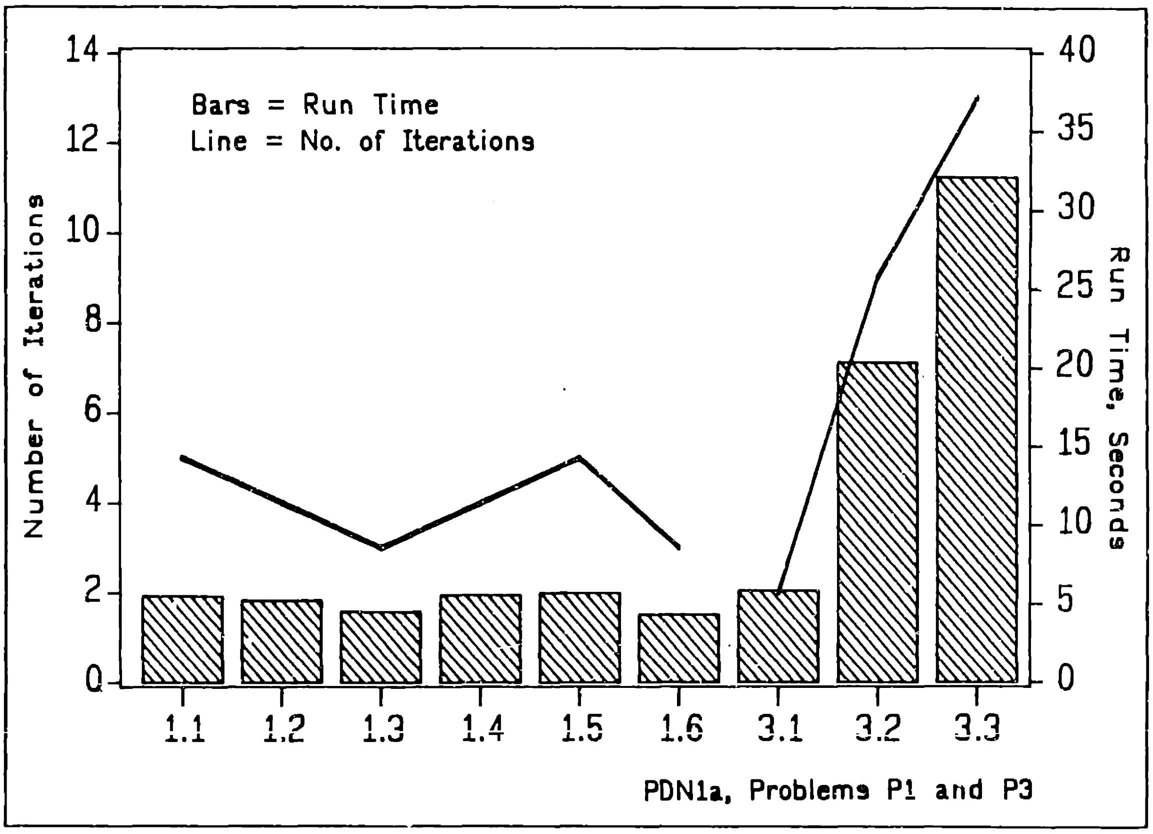

Figure 8.1: Run Time and Number of Iterations (P1 & P3)

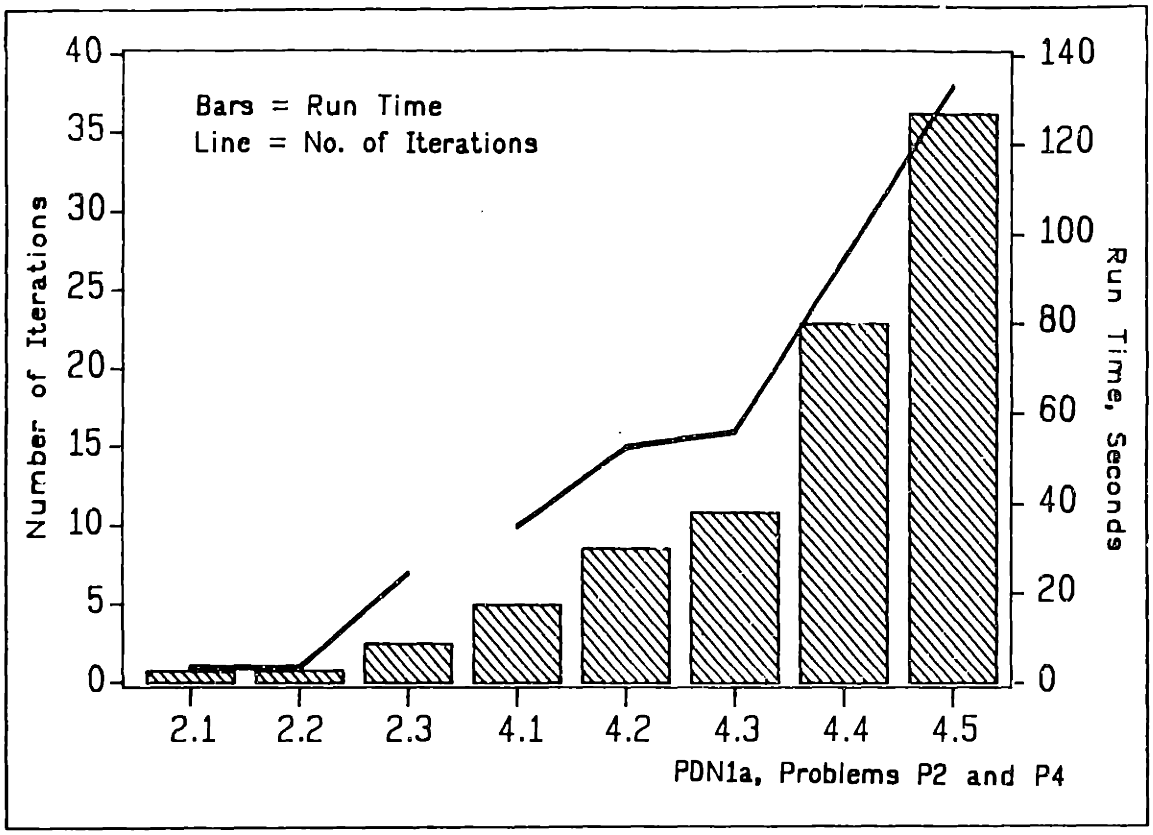

Figure 8.2: Run Time and Number of Iterations (P2 & P4)

8.2. Run Time vs. Total Demand

For problems P2, P3 and P4, total demand increases with problem number (Figures 8.3 and 8.4). For example, total demand is greater for problem P3.2 than for problem P3.1, just as total demand in problem P3.3 exceeds that in problem P3.2. This leads to the next general observation drawn from the plots of Figures 8.3 and 8.4-- total run time increases as total demand increases.

To explain this relationship, consider that as total demand increases, more of the available network capacity will be utilized. In

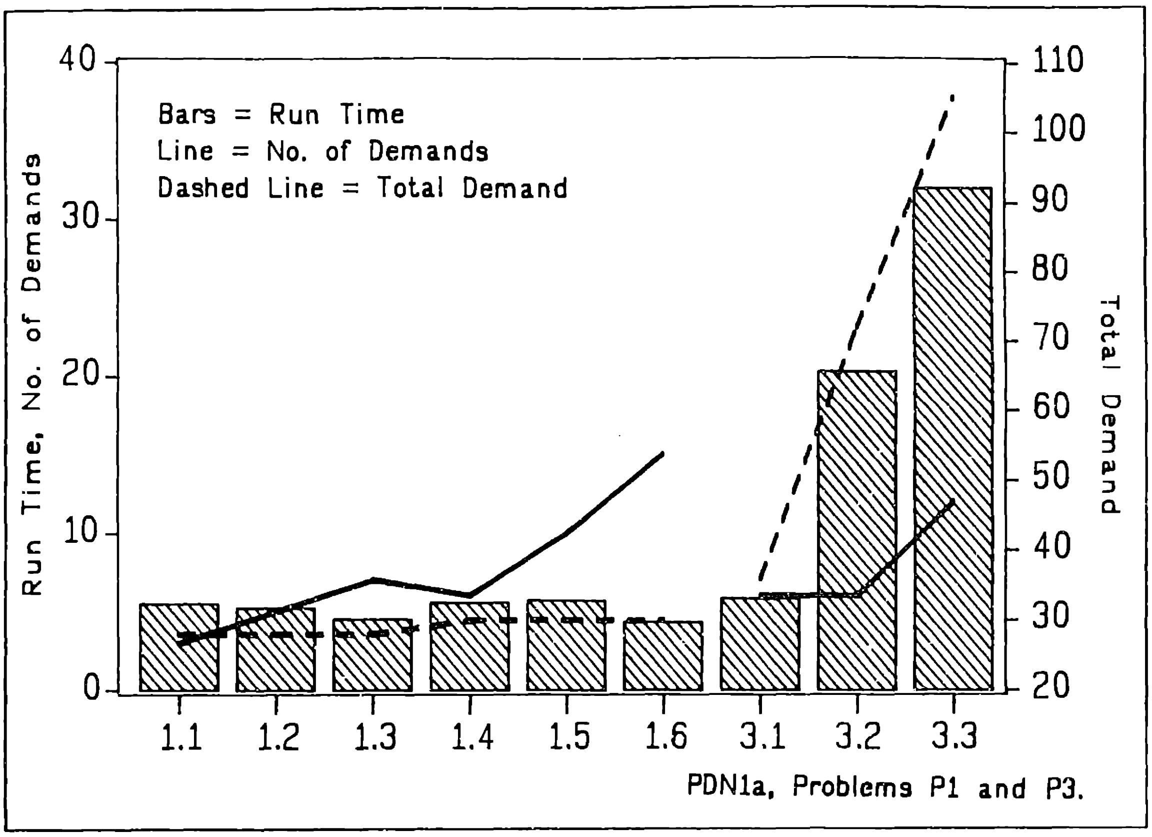

Figure 8.3: Run Time, Total Demand and Number of Demands (P1 & P3)

other words, more of the network paths will have assigned flow. As the number of paths with assigned flow increases, the number of dual price adjustments (or equivalently, the number of iterations) required in the PDN algorithms to equilibrate the lengths of these paths (so that all are shortest) increases. Hence, increases in total demand should result in increases in the number of iterations required to achieve optimality. Figure 8.5 verifies this conclusion by showing for problems P2, P3 and P4, an increase in the number of iterations as total demand increases. As previously discussed, the total run time

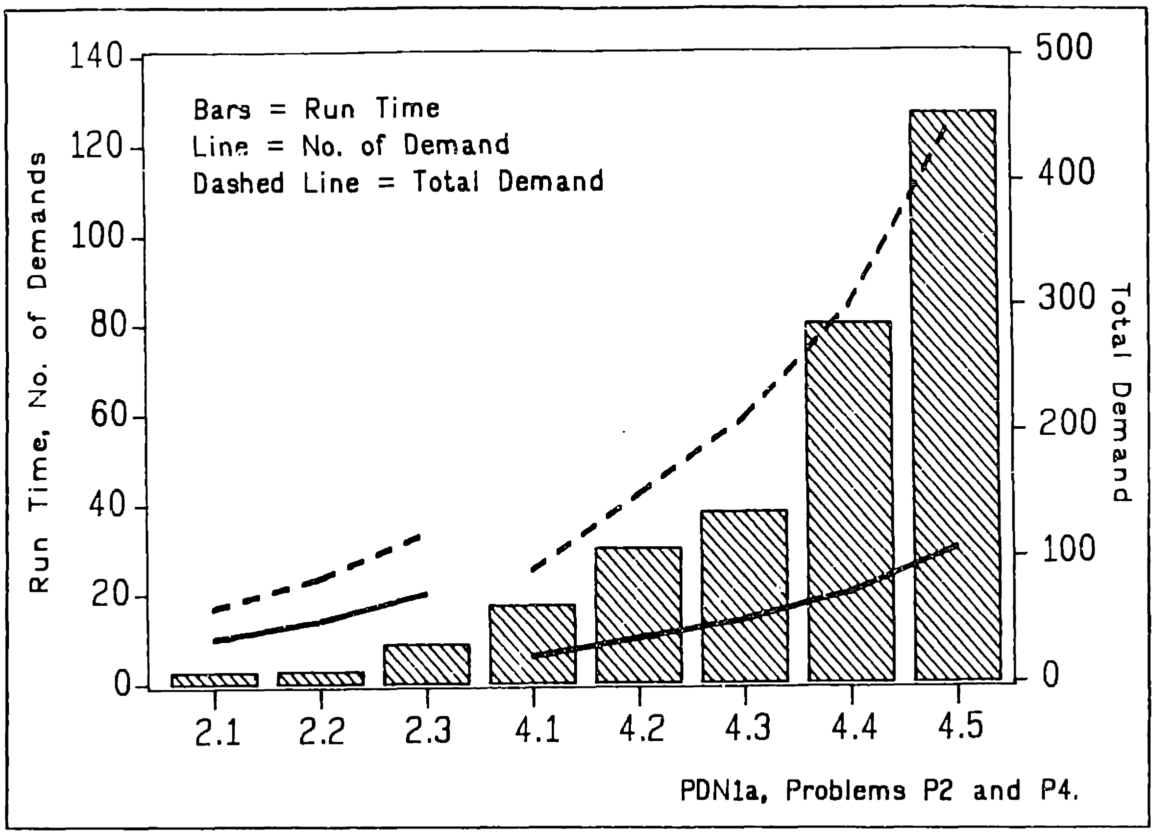

Figure 8.4: Run Time, Total Demand and Number of Demands (P2 & P4)

is strongly influenced by the number of iterations. Hence, the increase in run time resulting from increases in total demand is explained, at least in part, by the increased number of iterations performed to achieve optimality.

8.3 Run Time vs. Number of Demands

The increases in total run time with problem number for problems P4, are in part due to increases in dual times. In the dual adjustment step of the PDN algorithms, the optimal value of θ (equation [2.33]) is determined and then, the dual arc and node prices are

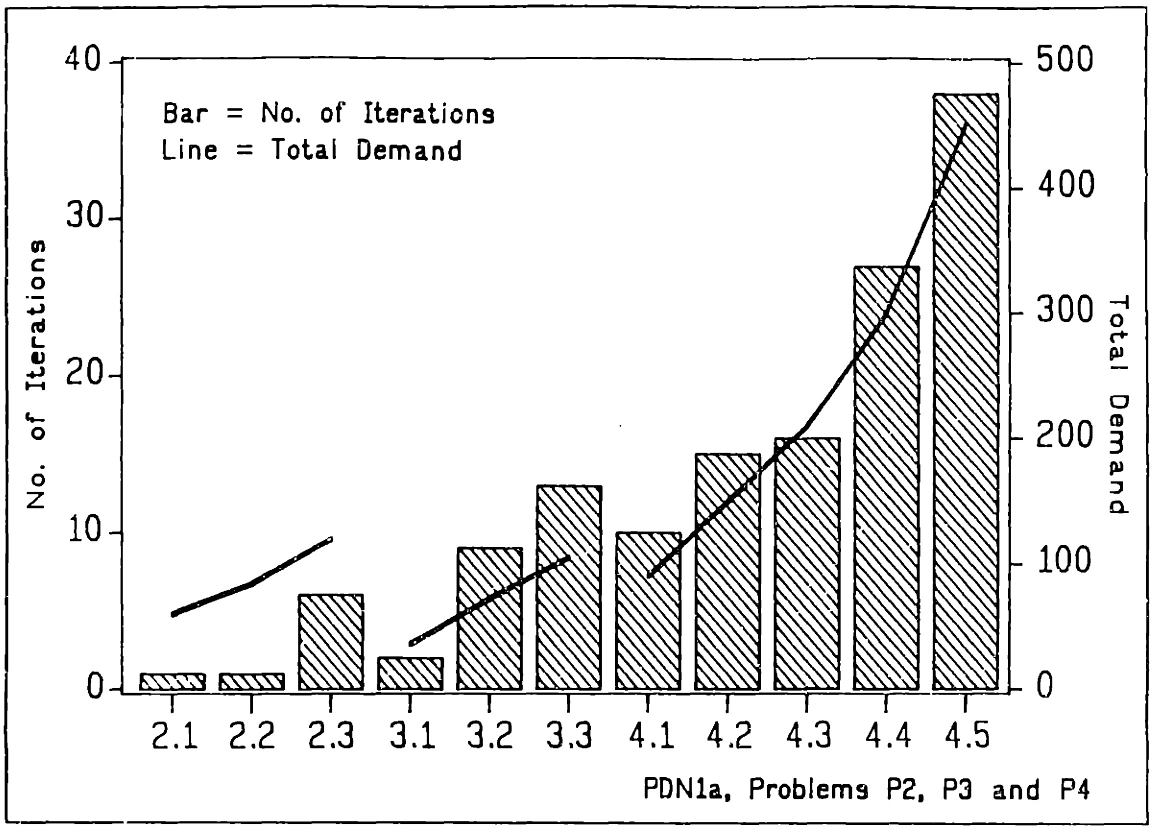

Figure 8.5: Number of Iterations and Total Demand

updated. These steps are achieved with label-correcting algorithms, one performed for each commodity. In problems P4, since the number of commodities increases with increasing problem number, there is a resulting increase in the number of label-correcting algorithms performed for each dual adjustment step. This means that the number of label-correcting algorithms performed per iteration increases as the number of commodities increases. Hence, as shown in Figure 8.6, the average dual time per iteration increases when moving from problem P4.1 up to problem P4.5. Contrast this with the corresponding pattern for Problem P3, also shown in Figure 8.6. Although the number of demands for problems P3 increases with problem number, the number of commodities (defined as trees) stays constant. In a PDN dual adjustment step, the label-correcting algorithms are performed for each commodity and not for each demand. Hence, the average dual time per iteration for problem P3 does not display the same increasing pattern as in problem P4.

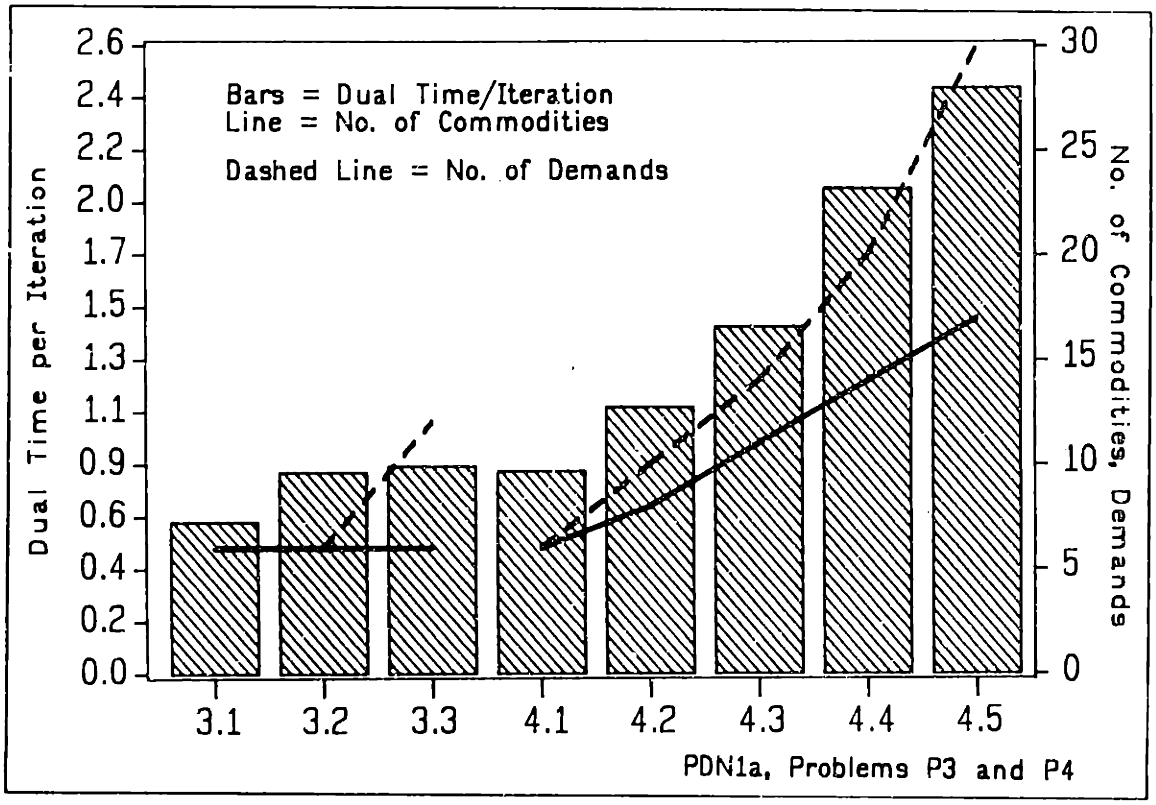

Figure 8.6: Dual Time Per Iteration and Number of Commodities

8.4 Run time vs. Average Demand Size

For problems 1.1 through 1.3 and problems 1.4 through 1.6, Figure

8.3 shows that a general pattern of decreased run times results when total demand is held constant and the number of demands is increased. This result can be explained by first observing that the average size of each demand decreases as the number of demands increases for a fixed total demand level. It is expected that as the number of demands increases for a fixed total demand level, flow will be more evenly distributed across the network and thus, the demand for arc capacity will be less concentrated. As a result, it is more likely that one demand will not have to split among paths but instead, will be fully assigned to a single path. Since each single path assignment can be simply determined in the PDN algorithm with a shortest path algorithm, the total time required to solve the RP problems is expected to decrease as the number of demands increases for a fixed demand level. Figure 8.7 supports this line of reasoning by showing a decrease in the PDN1a RP solution time as the average demand size decreases in problems P1.

The following conclusions are drawn from the preceding analysis:

i) algorithmic run time is expected to increase as the number of iterations increases;

ii) the number of iterations is expected to increase as total demand increases;

iii) RP solution time is expected to increase as the average size of each demand increases; and

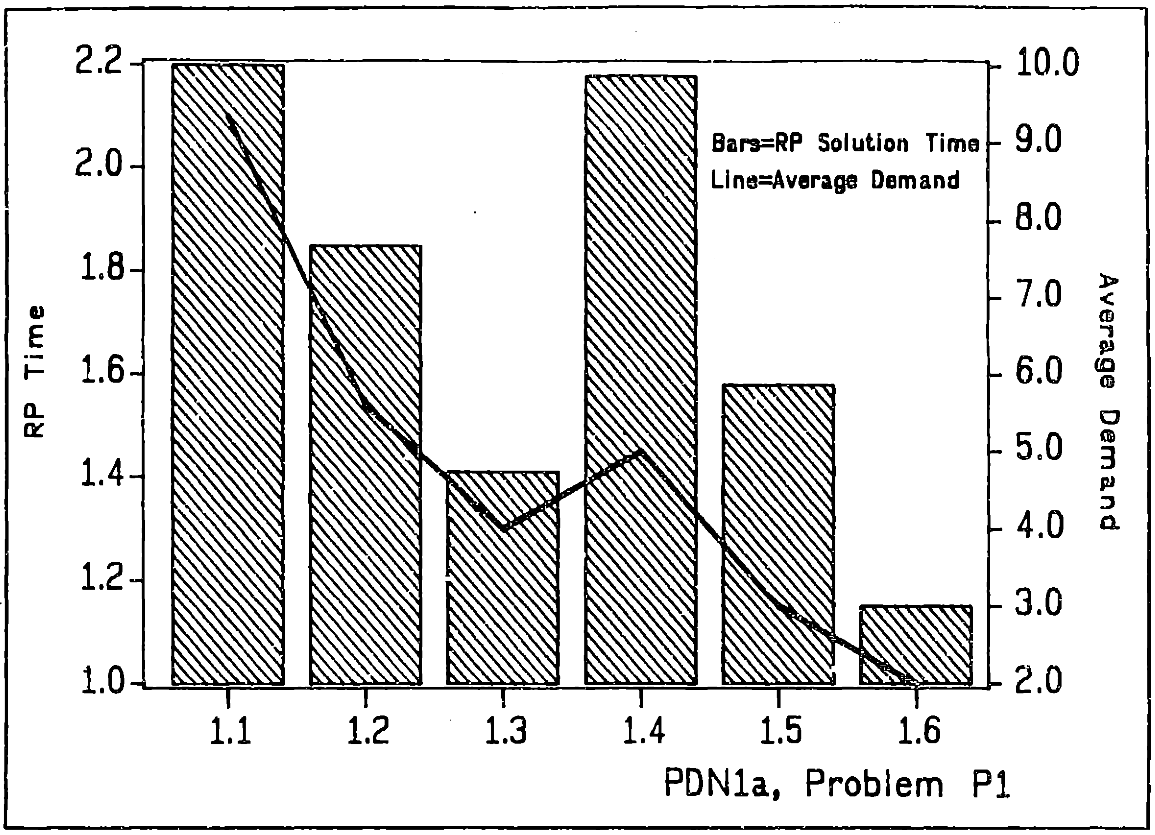

Figure 8.7: RP Solution Time and Average Demand

iv) dual time is expected to increase as the number of commodities increases.

Next, the effects of variations in the algorithmic solution strategy are examined.

8.5 Run Time vs. Flow Shift Criterion

The (a) and (b) versions of the PDN1 (PDN2) algorithmic implementations differ from one another only in the rules governing the flow shifts. Both the (a) and (b) versions shift flow to achieve a strict decrease in total infeasibility. However, unlike PDN1a (PDN2a), a flow shift in PDN1b (PDN2b) may result in an increase in infeasibility on a particular arc. In comparing the results of the two implementations, it was first observed that for each test problem, the same pattern of dual objective function ascent was achieved from one iteration to the next for both the (a) and (b) versions of PDN1 (PDN2). Thus, any possible difference between the (a) and (b) versions lies in the flow adjustments executed between dual ascents. Figure 8.8 shows that the (a) and (b) implementations of PDN1 (and of PDN2) are very similar in the total number of flow shifts required. As a result, the total RP times for the (a) and (b) implementations of PDN1 are not significantly different (Figure 8.9). The total RP solution times, however, are about 5% less for PDN2a than for PDN2b (Figure 8.9). This indicates that PDN2a is more efficient than PDN2b in finding "acceptable" flow shifts. This is counter-intuitive, however, since an "acceptable" flow shift is more strictly defined for the PDN2a case than the PDN2b case. However, since PDN2b is more "relaxed" in its flow shift criterion, it may be that the initial flow shifts reassign too much flow and must later be partially undone. As a result, the later flow shifts of PDN2b might be more involved (i.e. including more commodities) than those of PDN2a. This perhaps explains the slight increase in efficiency achieved by the PDN2a RP implementation over that of PDN2b.

The overall conclusion, however, is that no significant difference in the solution process or run times is achieved by altering the rules governing flow shifts.

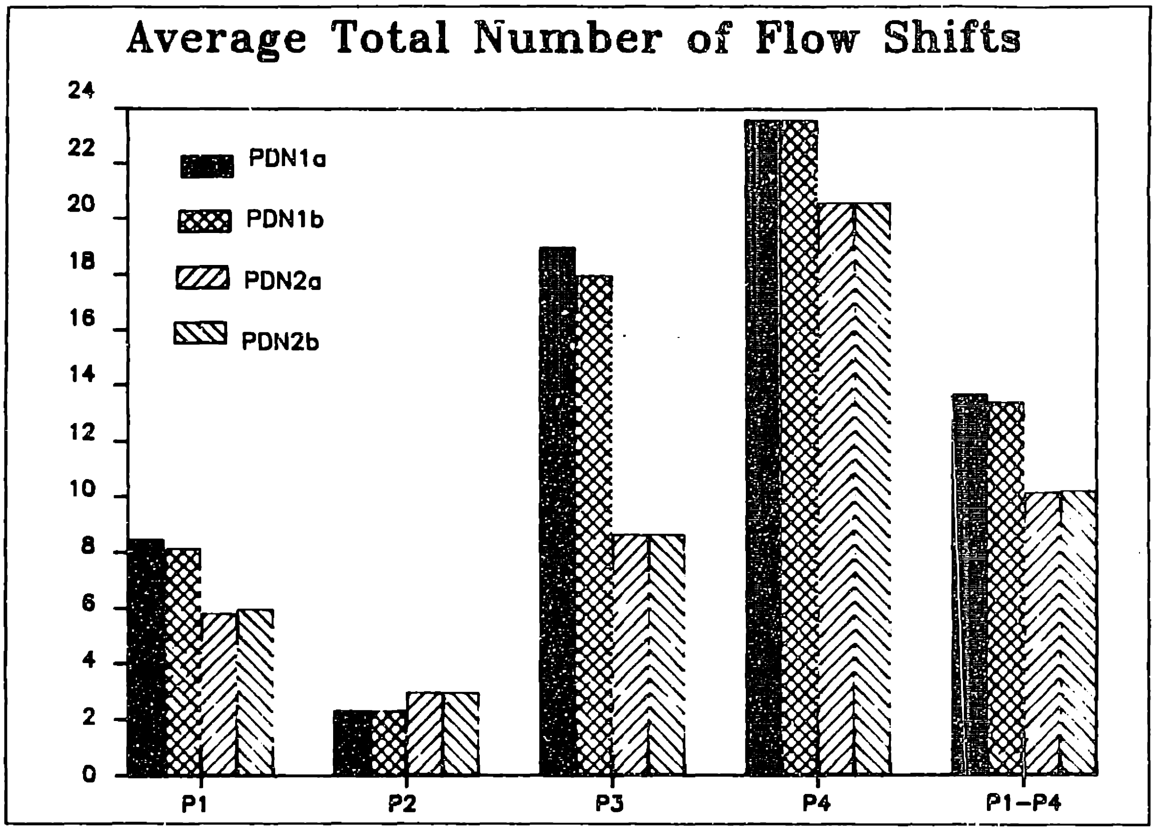

Figure 8.8: Average Total Number of Flow Shifts

Next, in Sections 8.6 through 8.10, the performance of each algorithmic implementation is evaluated by comparing among implementations the number of iterations; initialization time; RP solution time; dual time; and total run time.

8.6 LP Solution Methodology

In the LP solution Methodology, as described in Chapter 7, the linear program for each test problem is solved using MINOS. To reduce

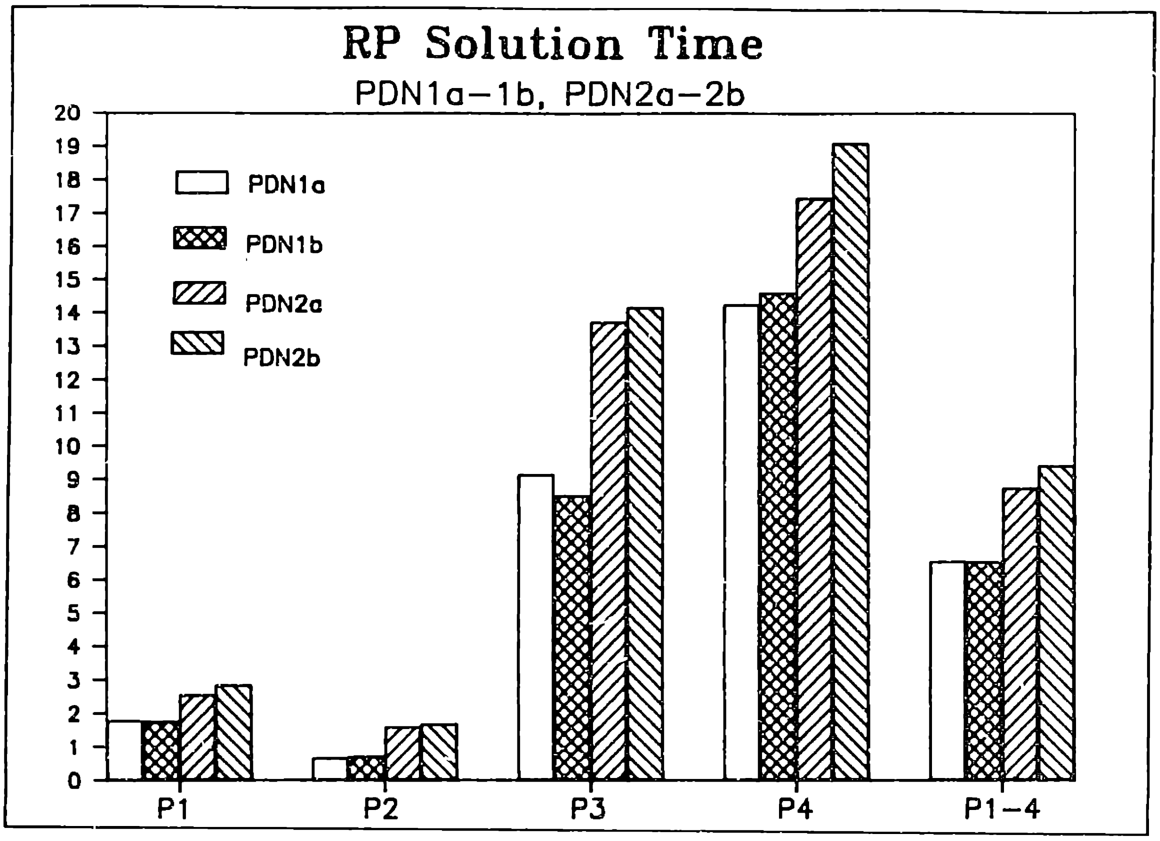

Figure 8.9: PDN RP Solution Time

(sometimes substantially) the number of commodities and hence, the size of the corresponding LP, commodities are defined by the tree representation rather than the path representation. Even with this more efficient problem formulation, the LP solution method is not competitive with the other solution methods in terms of run times.

As shown in Figure 8.10, all other solution methods (i.e. the DW, PD and PDN implementations) averaged together, outperform the LP implementation. Averaging over problems P1, P2 and P3, the run time for the LP solution method is about 100 times greater than that for

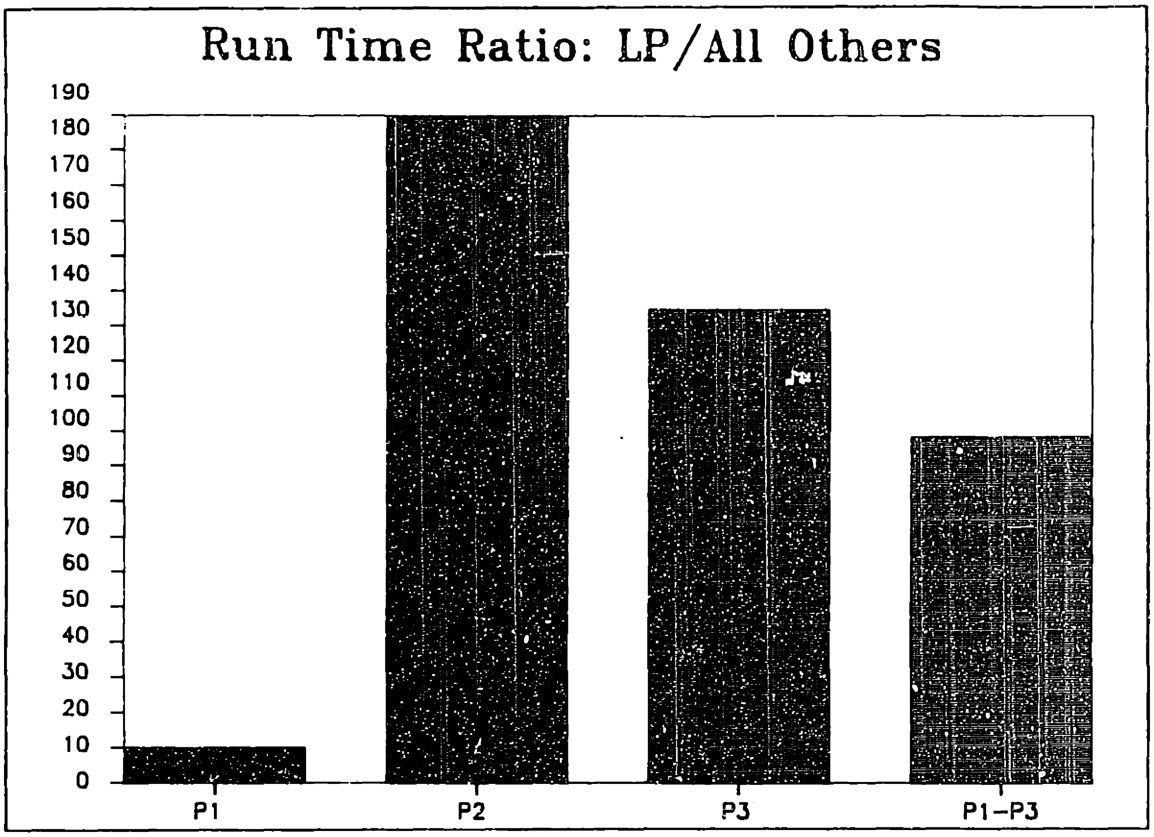

Figure 8.10: Run Time Ratio: LP vs. All Others

the other methodologies averaged together. (This rather sharp contrast in run times between the LP and all other solution methods actually **underestimates** the "true" difference. The average run times shown in Figure 8.10 do not include results for problems P4 which the LP solution method did not solve because too many iterations were required.) Figure 8.10 shows that as the number of arcs or nodes in the network is increased (problems P3) or as the number of commodities is increased (problems P2), run times of the LP solution methodology grow much faster than do the corresponding run times for the other solution methodologies. The LP solution methodology is particularly sensitive to problem size because expected solution times grow with the size of the linear program. The size of the linear program, in turn, is determined by the number of nodes, the number of commodities and the number of arcs in the problem to be solved. Specifically:

$$m = N*K + C;$$
$$n = A*K. \qquad [8.1]$$

where:

m: number of rows in the LP;

n: number of columns in the LP;

N: number of nodes in the test problem;

K: number of commodities in the test problem;

C: number of arcs with finite capacity in the test problem;

A: total number of arcs in the test problem.

It should not be surprising that the LP solution method is outperformed by the DW, PD and PDN solution methodologies. Unlike the other methods, the LP solution method exploits neither the block-angular nor the network structure of the CMCF problem.

The remainder of this chapter will focus on the more specialized solution methods.

8.7 PDN Solution Methodology

As stated in Chapter 7, the PDN1 and PDN2 implementations differ in the formulation of the RP problem. PDN1 formulates the RP problem using equations [2.16] - [2.20], whereas PDN2 formulates the RP problem using equations [7.1] - [7.5].

8.7.1 PDN1 vs. PDN2 Implementations

As shown in Figure 8.11, the total run times and the individual component times (averaged over all problems P1 through P4) are essen tially equal for PDN1a and PDN1b, and similarly for PDN2a and PDN2b. However, between the PDN1 and PDN2 implementations, there is some

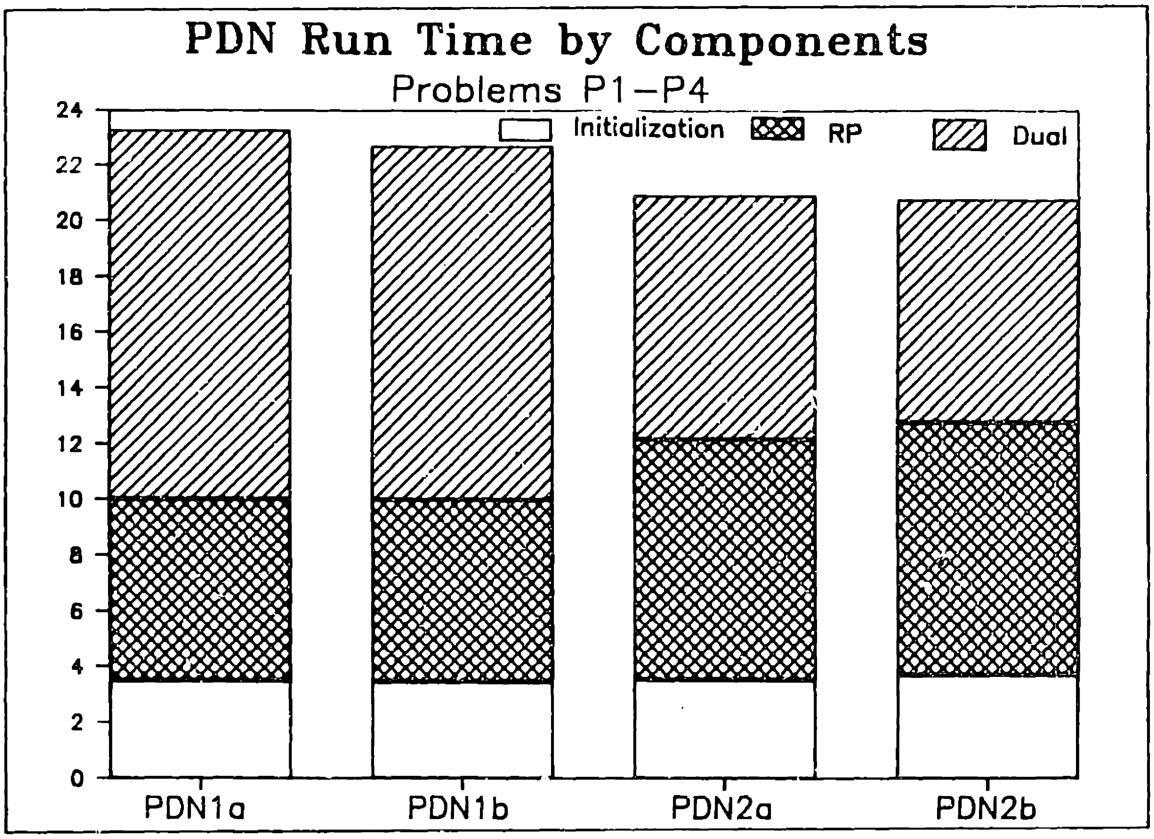

Figure 8.11: Run Time by Components

variation in the these times. Although, the PDN2 and PDN1 implementations both require the same average number of iterations to achieve optimality, the PDN2 implementations outperform the PDN1 implementations with respect to total run time. A component by component comparison shows that the initialization times are the same for each implementation; RP solution times are less for PDN1 than for PDN2; and the dual times are greater for PDN1 than for PDN2.

8.7.2 PDN1 vs. PDN2: Number of Iterations

As shown in Figure 8.12, the PDN1 and PDN2 algorithmic implementa-

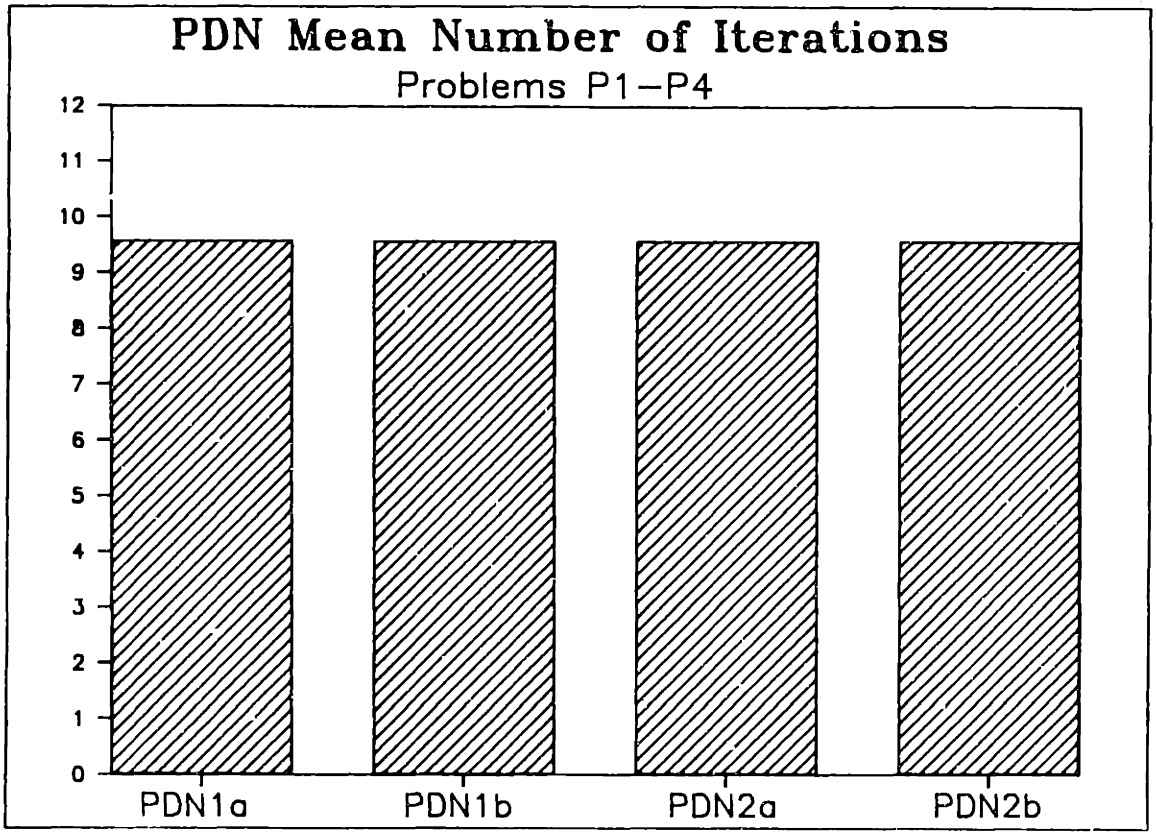

Figure 8.12: PDN Mean Number of Iterations

tions require essentially the same average number of iterations to achieve optimality. However, it is interesting to note that the dual objective function value achieved at each iteration, differs for the two methods. Figure 8.13, for example, shows for Problem 4.3 (a representative case) the dual objective function value achieved at each iteration of the PDN1a and PDN2a implementations. Although both algorithms require the same number of iterations to solve the problem, the path followed in attaining the optimal solution is different for the two methods.

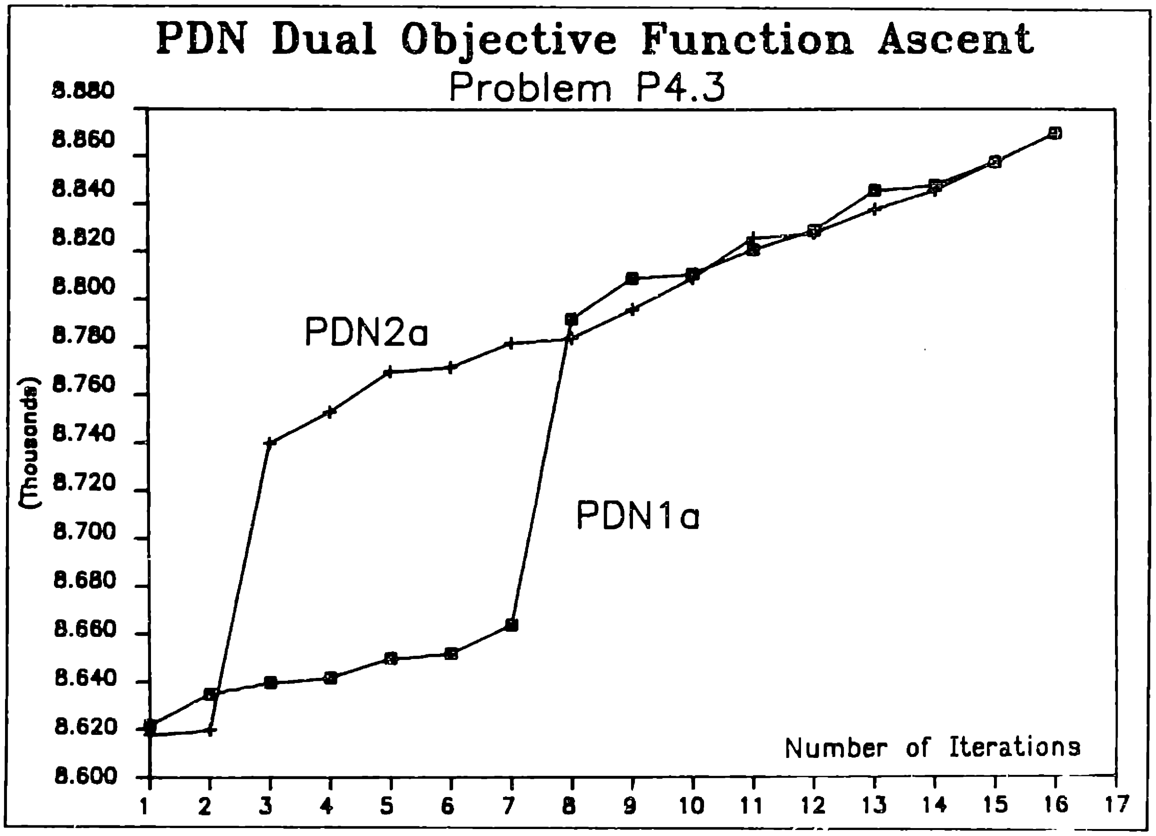

Figure 8.13: PDN Dual Objective Function Ascent

8.7.3 PDN1 vs. PDN2: Initialization Times

The equivalence of the PDN1 and PDN2 initialization times (shown in Figure 8.11) is expected since the initialization components are exactly the same for each of the PDN implementations.

8.7.4 PDN1 vs. PDN2: Dual Times

Dual adjustment steps are executed when a flow adjustment search fails to find an acceptable flow shift. In the PDN algorithm, the search for a flow shift may involve only one commodity or, if the domino-effect is encountered, may involve several or all commodities. In order to determine and set the new dual prices, it is necessary to run the label-correcting algorithms only for those commodities involved in the failed flow shift search. In all of the PDN algorithms, the dual adjustment step implementation is the same. Thus, any difference in the average time for a dual adjustment step between PDN1 and PDN2 is attributed (in part anyway) to the difference in the number of label-correcting algorithms performed. In the PDN2 implementations, the average time for a dual adjustment step, or equivalently, the average dual time per iteration is less than the corresponding time in the PDN1 implementations (Figure 8.14). This means that, on average, the RP formulation of PDN2, as compared to that of PDN1, results in the involvement of fewer commodities in the flow shift search.

8.7.5 PDN1 vs. PDN2: RP Solution Times

The average number of flow shifts performed in the PDN1 implemen-

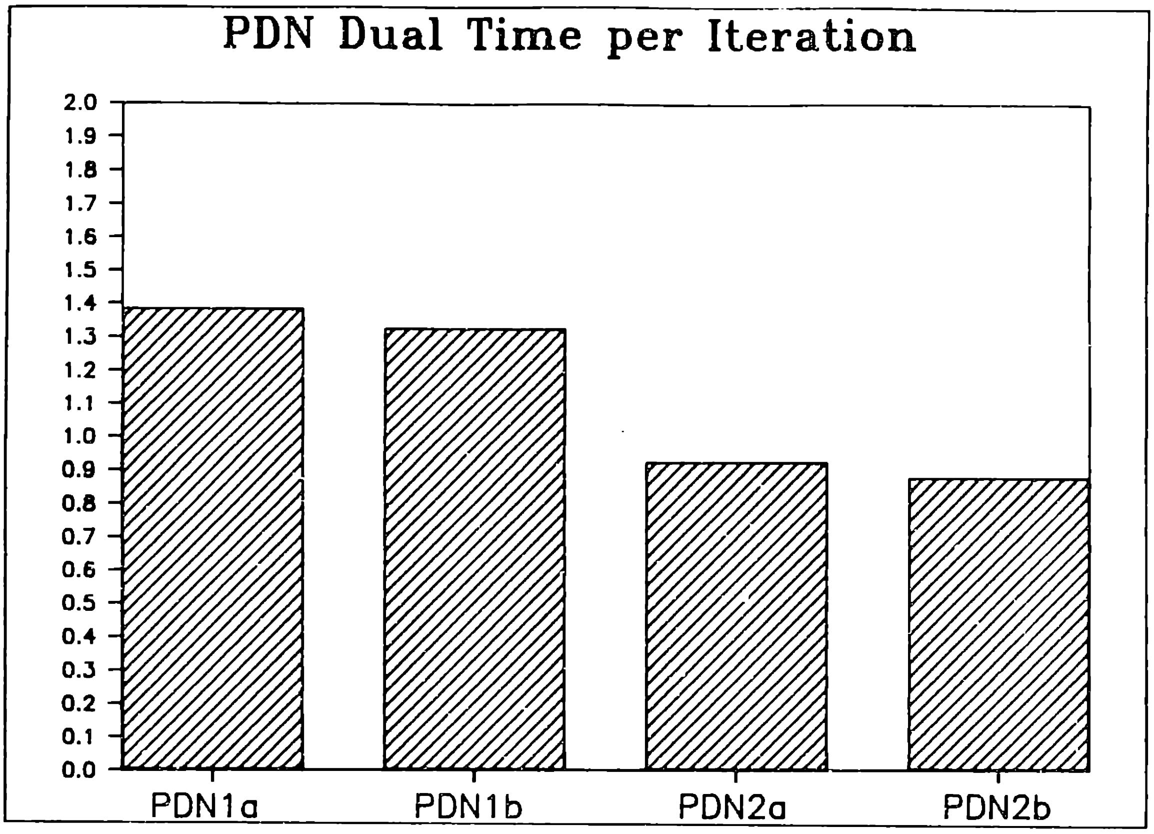

Figure 8.14: PDN Dual Time per Iteration

tations exceeds the number performed in the PDN2 implementations (Figure 8.8). Even so, the total RP solution time is less for the PDN1 implementations than for the PDN2 implementations. Hence, the average time per flow shift in PDN1 is less than the average time in PDN2. This indicates that the search for a flow shift is taking longer in the PDN2 implementation than in the PDN1 implementation. The longer search process of PDN2 indicates that either:

i) more commodities are involved in a typical flow shift for PDN2 than for PDN1; or

ii) a greater portion of the network is searched in PDN2 than in PDN1.

As described in Section 8.7.4, the PDN1 and PDN2 dual times do not support the first hypothesis above. Since the average dual time per iteration is less for PDN2 than for PDN1, fewer commodities are involved in the flow shift search of PDN2 than that of PDN1. Since the first hypothesis above is rejected, it follows that a greater portion of the network is examined in the PDN2 search than in the PDN1 search. To understand why, compare typical PDN1 and PDN2 search processes.

In the PDN2 search, an objective is to remove flow from the artificial origin-to-destination arcs for each commodity. In the search for such a flow shift, a cycle must be found which contains a path from the origin node to the destination node of the commodity under consideration. Hence, the entire network must be examined in the search for this origin to destination path. Contrast this search process with that of the PDN1 algorithm. In the PDN1 algorithm, the objective is to remove flow from an OCarc or add flow to an UCarc. Again, to achieve this goal, a cycle must be found which contains a path, this time, from the origin node to the destination node of the selected problem arc. Hence, it may not be necessary to search the entire network, but rather only that small portion of the network centered about the problem arc.

Thus, the greater RP solution time for the PDN2 implementations as compared to the PDN1 implementations is explained by the fact that a greater portion of the restricted network is involved in the search for flow shifts.

8.7.6 PDN1 vs. PDN2: Total Run Time

Table 8.1 compares the performance of the PDN algorithmic implementations. The PDN1 and PDN2 methods follow a different path in achieving the optimal solution but both methods execute approximately the same number of iterations in attaining optimality. PDN2, as compared to PDN1, determines the size of the dual adjustment as a function of fewer commodities on average and thus, spends less time performing the dual adjustment step at each iteration. The PDN1 implementation, however, spends less time in finding appropriate flow

	PDN1 vs. PDN2
Total Run Time	>
Average Number of Iterations	≈
Average Initialization Time	-
Average RP Solution Time/Iteration	<
Average Dual Time/Iteration	>

Table 8.1: PDN1 vs. PDN2

shifts because it more severely restricts its network search. The amount of time saved by PDN1 in solving the RP subproblems, however, does not compensate for the extra time spent in the dual adjustment steps. As a result, for the set of test problems, the overall average run time for the PDN2 implementations is slightly better than that for the PDN1 implementations (Figure 8.11).

8.8 PD Solution Methodology

The PD solution methodclogy, as explained in Chapter 7, implements the primal-dual method (Chapter 2) and uses the simplex algorithm to solve the RP subproblems. The PD1 and PD2 implementations differ only in the formulation of the RP problems. In the PD1 implementation, the RP problems are formulated using equations [2.16] - [2.20] whereas, in the PD2 implementation, the RP problems are formulated using equations [7.1] - [7.5].

8.8.1 PD1 vs. PD2 Implementations

The total run times and the individual component times (averaged over all problems P1 through P4) for the PD1 and PD2 implementations are compared in Figure 8.15. Overall, the PD2 implementation solves the test problems faster than does the PD1 implementation. A component by component comparison reveals that the initialization and dual times per iteration are greater for PD1 than for PD2 and their RP solution times per iteration are essentially equal.

8.8.2 PD1 vs. PD2: Number of Iterations

To solve the test problems, the PD2 implementations require an average of approximately 7% fewer iterations than the PD1 implementa-

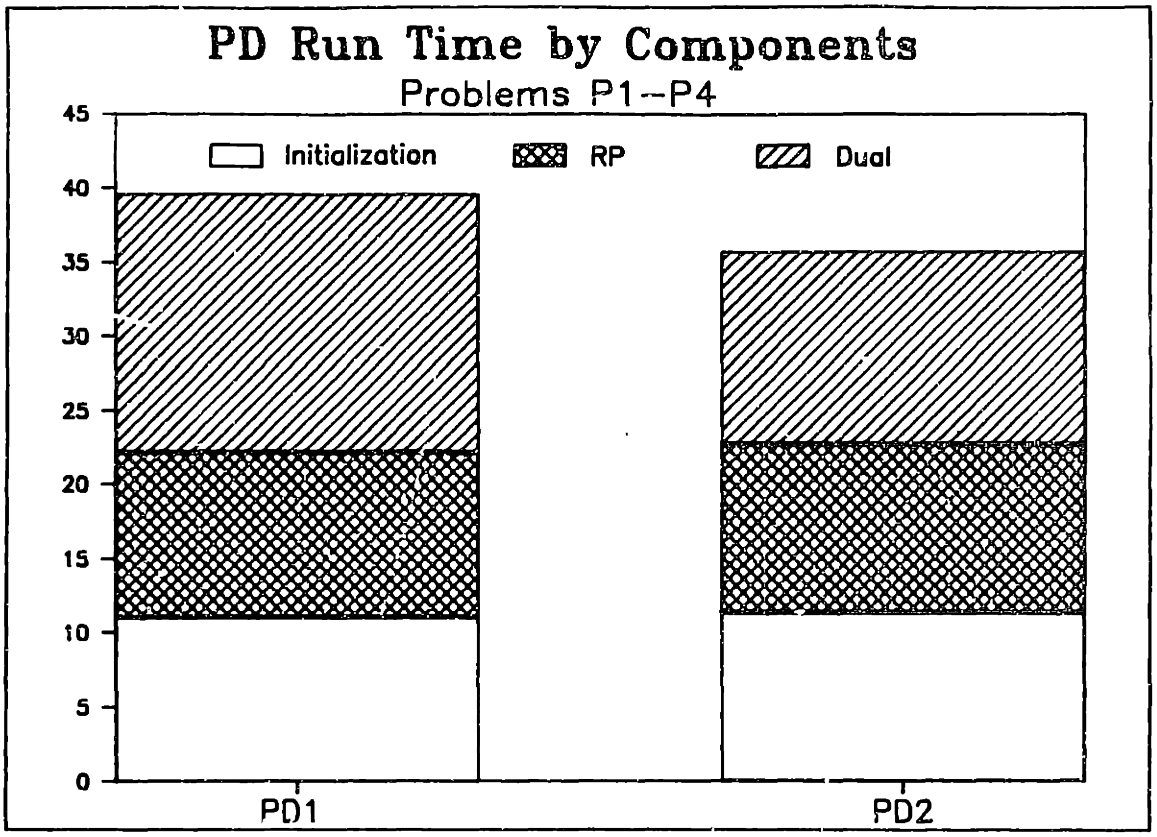

Figure 8.15: PD Run Time by Components

tions, as indicated by Figure 8.16. Figure 8.17 shows, for Problem P1.1, the tracing of the dual objective function value at each iteration of the PD1 and PD2 methods. This particular example highlights the differences in the solution processes for the two algorithmic implementations. The PD2 implementation achieves steeper ascent of the objective function value than does PD1. PD2 tends to follow a repeating sequence where steep ascent followed by no ascent is achieved. (The "no-ascent" step occurs when another equal length "shortest" path which has not yet been added into the simplex tableau is found.) Unlike PD2, PD1 tends to achieve ascent, however slight, at each iteration. This phenomenon is best explained through an example.

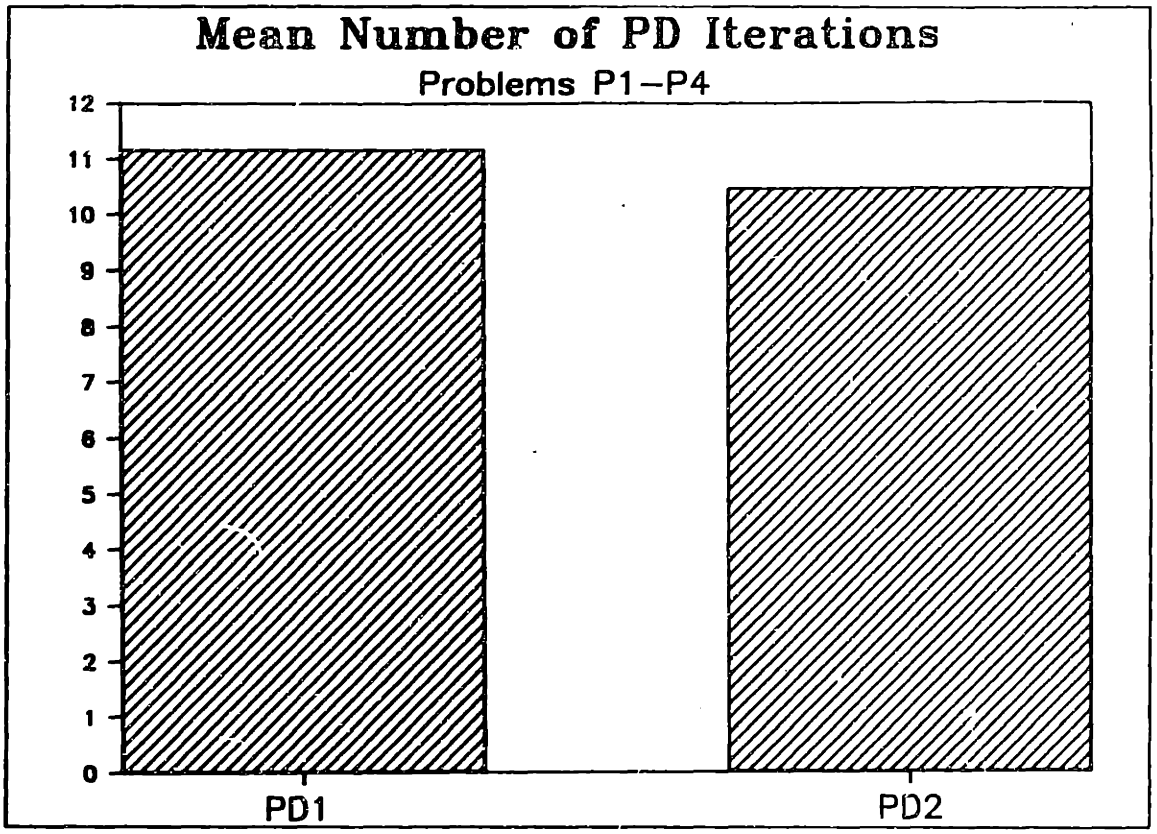

Figure 8.16: Mean Number of PD Iterations

Consider the network, for a single commodity k, depicted in Figure 8.18. In the PD1 implementation, (Figure 8.19), the initialization process assigns, without regard to arc capacities, all of commodity k to the origin-to-destination path denoted p1. This assignment results in the over-capacitation of arcs a1, a2 and a3. By complementary slackness conditions for the RP problem (equations [5.11a]), the RP

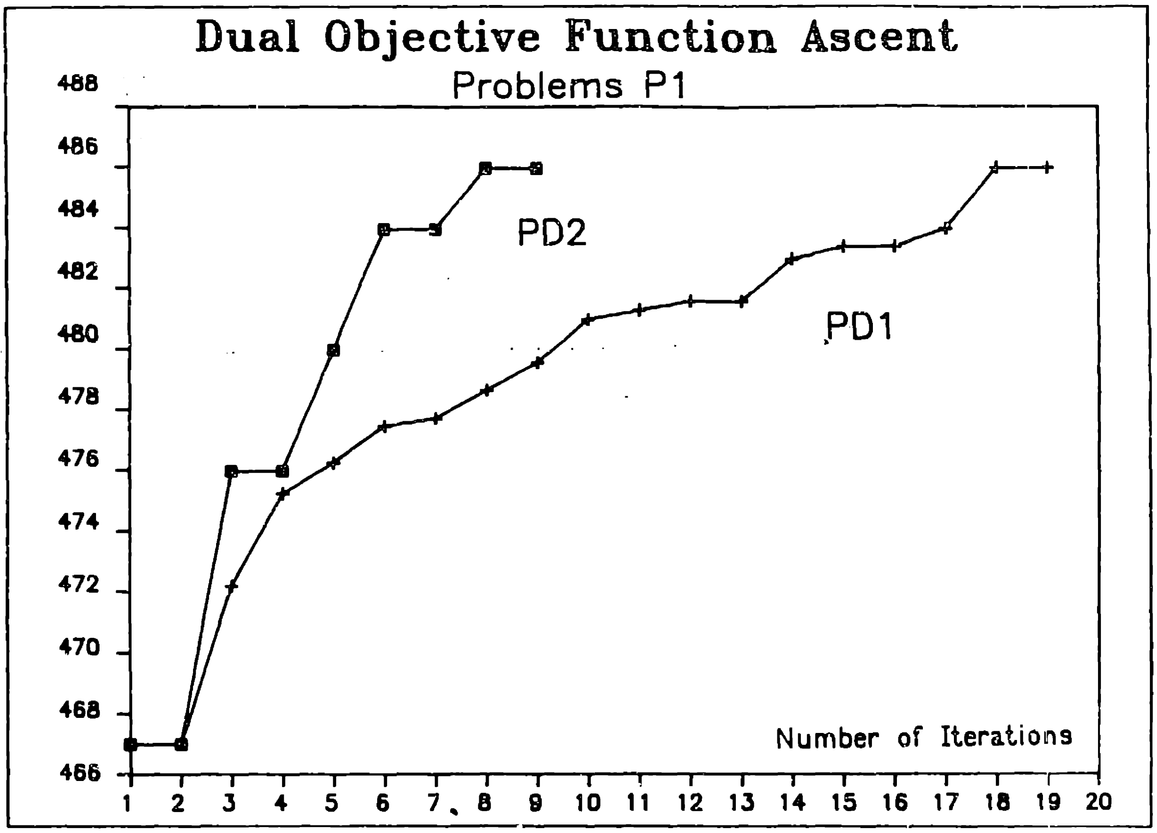

Figure 8.17: PD Dual Objective Function Ascent

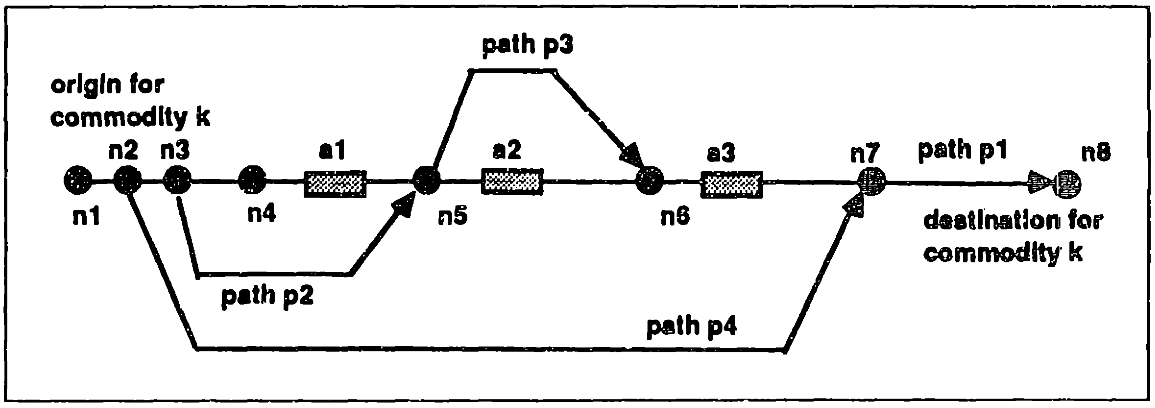

Figure 8.18: Example Network

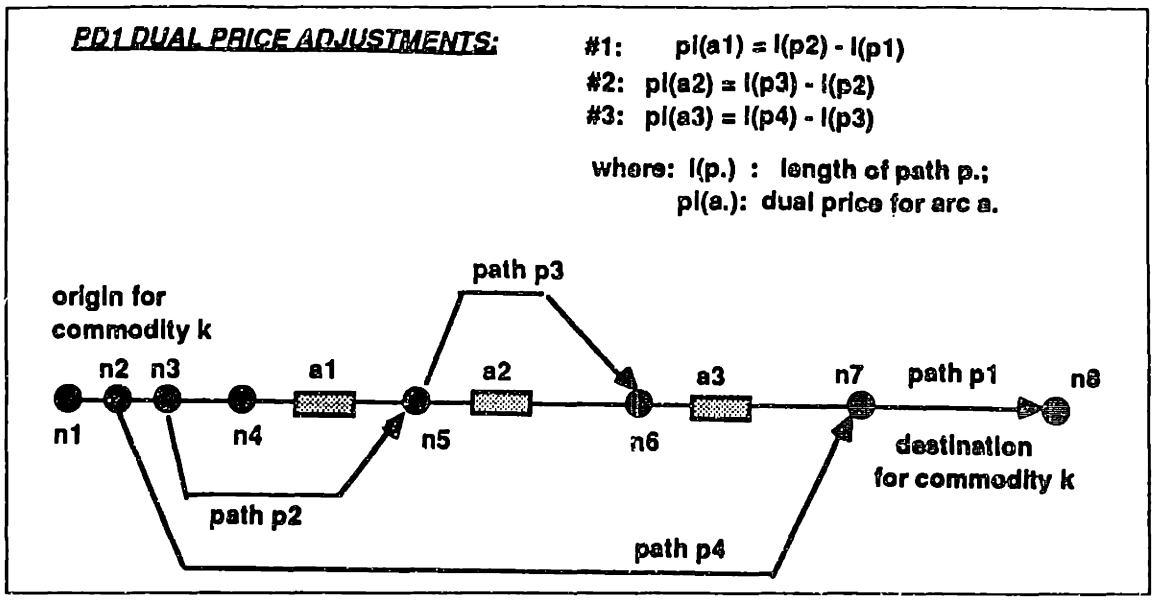
PD1 DUAL PRICE ADJUSTMENTS:
#1: pl(a1) = l(p2) - l(p1)
#2: pl(a2) = l(p3) - l(p2)
#3: pl(a3) = l(p4) - l(p3)
where: l(p.) : length of path p.;
pl(a.): dual price for arc a.
path p3
origin for commodity k
n2 n3
a1
a2
a3
n7
path p1
n8
n1
n4
n5
n6
destination for commodity k
path p2
path p4

Figure 8.19: PD1 Solution Approach

solution sets the $\Delta\pi$ dual prices for each of these arcs to +1. Next, the dual price adjustment step finds that arc a1 determines the size of the price adjustment such that the length of path p1 is increased to that of the next shortest path-- path p2. The next iterations of PD1 are as follows:

i) arc a2 determines the size of the dual price adjustment and the length of paths p1 and p2 are increased up to the length of path p3; and

ii) arc a3 determines the size of the dual price adjustment and the lengths of paths p1, p2 and p3 are increased up to the length of path p4.

Thus, in PD1, three iterations are required to increase the length of the admissible paths up to the length of path p4.

Compare the PD1 solution process with the PD2 solution process. In PD2 (Figure 8.20), the initial flow assignment capacitates arc a3 and assigns the excess flow of commodity k to the artificial origin to destination arc, denoted a0. In solving the resulting RP problem, the dual price on arc a3 is set to +1. Then, the dual adjustment step determines the size of the price adjustment such that path p1 (as well as paths p2 and p3) are increased in length up to that of path p4. Hence, PD2 accomplishes in one iteration the work performed in three iterations of PD1.

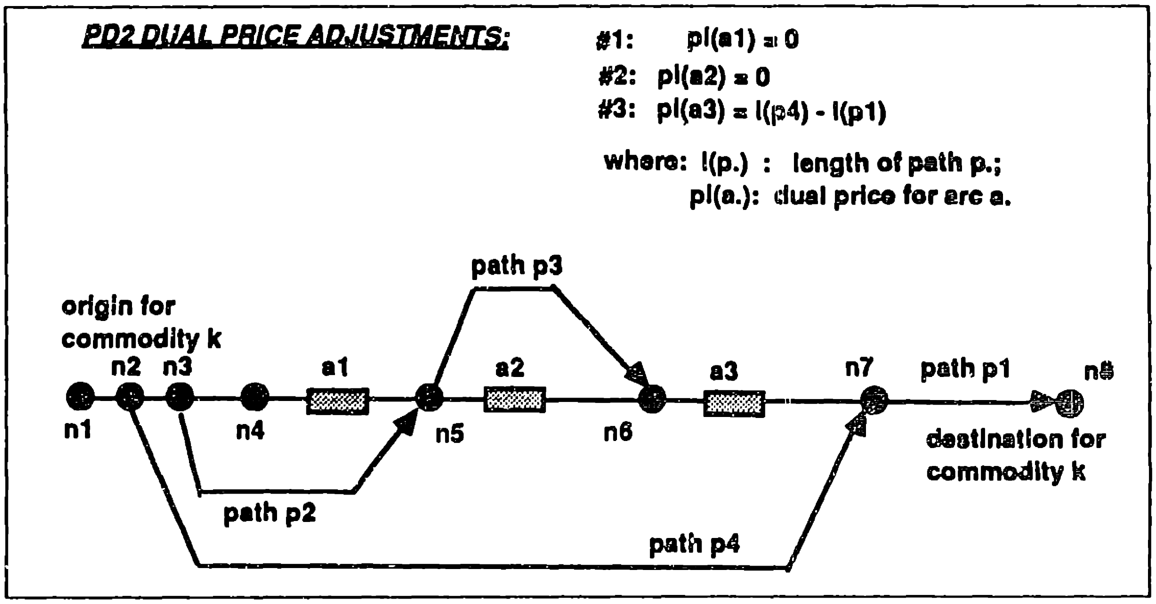
PD2 DUAL PRICE ADJUSTMENTS:
#1: pl(a1) = 0
#2: pl(a2) = 0
#3: pl(a3) = l(p4) - l(p1)
where: l(p.) : length of path p.;
pl(a.): dual price for arc a.
path p3
origin for commodity k
n2 n3
a1
a2
a3
n7
path p1
n8
n1
n4
n5
n6
destination for commodity k
path p2
path p4

Figure 8.20: PD2 Solution Approach

The above example shows that increased efficiency of the PD2 algorithm is achieved (at least partially) because the RP formulation of

PD1 results in more non-zero $\Delta\pi$ arc prices than does the RP formulation of PD2. When more arcs have non-zero $\Delta\pi$ values, more alternative paths are located and hence, more algorithmic iterations are performed.

8.8.3 PD1 vs. PD2: Initialization Times

The average initialization time for the PD1 algorithm is 22% greater than the initialization time for the PD2 algorithm. In the initialization step, both PD algorithms read in the input data and set up the initial simplex tableau. The number of artificial variables in the PD1 formulation (i.e. columns in the simplex tableau) roughly doubles the number in the PD2 formulation. It is this difference which accounts for the discrepancies in the initialization times for the two implementations.

8.8.4 PD1 vs. PD2: Dual Times

As with the PDN algorithmic implementations, the dual adjustment step in the PD1 and the PD2 algorithms are exactly the same. However, unlike the PDN implementations, in the PD implementations, the RP problem is solved to optimality with the simplex algorithm. Hence, each commodity is involved in the so-called "failed flow adjustment step" and thus, the label-correcting algorithms must be run for every commodity. Thus, the dual time per iteration should be the same for the PD1 and PD2 algorithms. The average dual time per iteration for the PD1 algorithm, however, is approximately 25% greater than that for the PD2 algorithm. To explain this difference, the label-correcting algorithm determining the optimal value of θ must be more explicitly described.

Equation [2.33] shows that the optimal value of θ is calculated by performing a calculation for each path p in the set α, where $\alpha = \{p | p \in (P^k), -\Sigma_{ij \in A} \Delta\pi_{ij}\delta_{ij}^{p,k} + \Delta\sigma^k - \gamma_p^k > 0; \forall k \in K\}$. Let ρ_p^k represent the amount by which path p exceeds the shortest path length for commodity k. Then, to calculate θ, the path with associated positive γ value and with the minimum ρ to γ ratio must be determined. The label-correcting algorithm accomplishes this task by finding for each commodity and for each possible positive γ value, the path of minimum length having that γ value. The θ computation can then follow directly using these paired ρ and γ values to determine the minimum ρ/γ ratio. Consider applying this algorithm to the examples in Figures 8.19 and 8.20.

To perform the θ calculation for the PD1 implementation, since $\Delta\sigma$ equals 3, the minimum length paths with γ values of 0, 1 and 2 have to be determined. (Path n1-n2-n7-n8 has $\gamma = 0$; path n1-n2-n3-n4-n6-n7-n8 has $\gamma = 1$; etc.) In the PD2 implementation however, $\Delta\sigma$ equals 1 and hence, the only computation required is the determination of the minimum length path with γ value of 0. Note that for problems with more than one commodity, these minimum length paths must be computed for every commodity and for each distinct positive γ value.

Thus, as a result of the RP formulation of PD1, as compared to that of PD2, additional work may be required to determine the optimal θ value. This added work results in longer dual times per iteration in the PD1 implementation than in the PD2 implementation.

8.8.5 PD1 vs. PD2: RP Solution Times

The RP time per iteration for PD2 and PD1 are essentially equal. This relationship is explained in Figure 8.21 which shows the average number of columns added to the simplex tableau at each iteration. Approximately the same number of columns are added and about the same

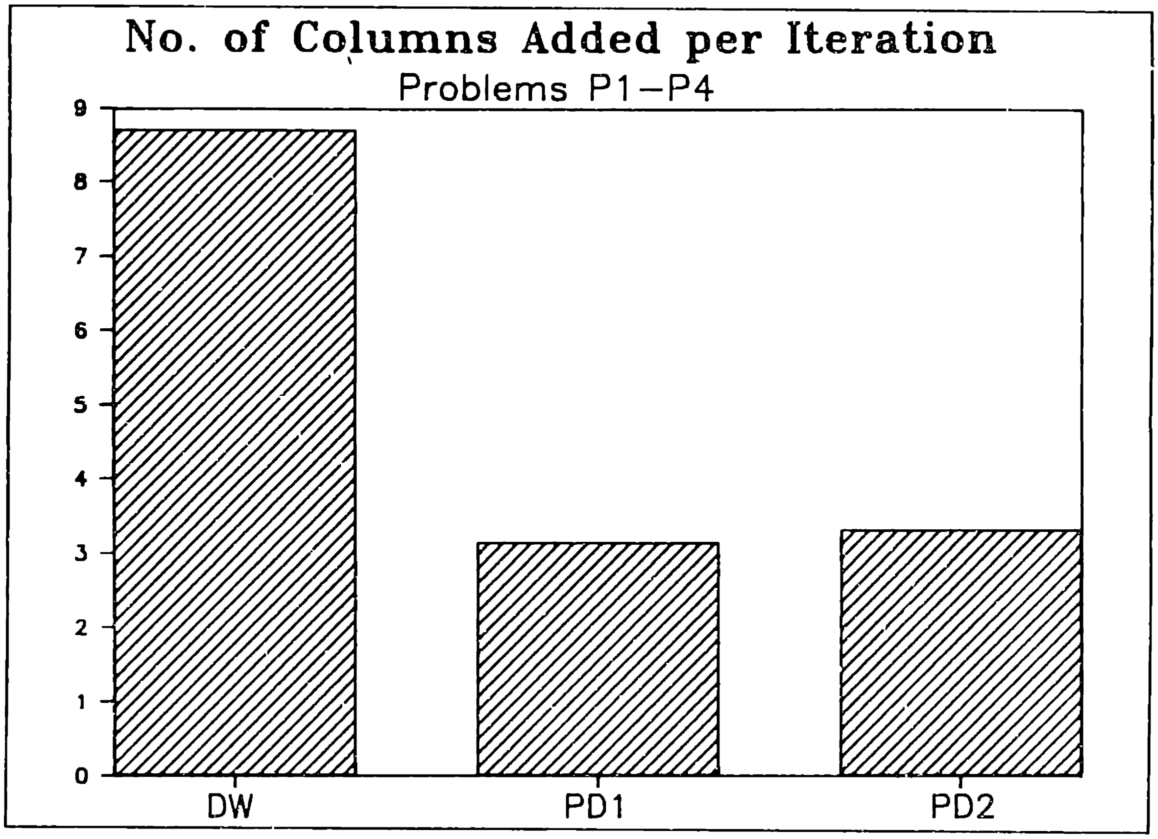

Figure 8.21: Number of Columns Added per Iteration

number of pivots occur per iteration in both PD1 and PD2. Consequently, the average RP time per iteration in the PD1 algorithm is about equal to that for the PD2 algorithm.

8.8.6 PD1 vs. PD2: Total Run Time

Table 8.2 summarizes the performance of the PD algorithmic implementations. As compared to the PD1 algorithm, the PD2 algorithm achieves optimality with fewer iterations because its RP formulation results in more efficient $\Delta\pi$ arc price assignments. Furthermore, both the average initialization time and the average dual time per iteration are shorter for PD2 than for PD1. The difference in the average

	PD1 vs. PD2
Total Run Time	>
Average Number of Iterations	>
Average Initialization Time	>
Average RP Solution Time/Iteration	≈
Average Dual Time/Iteration	>

Table 8.2: PD1 vs. PD2

dual time per iteration for the two implementations is, again, attributed to the more efficient $\Delta\pi$ arc price assignments resulting from the PD2 formulation as compared to those resulting from the PD1 formulation. The end result, is that the average total run time for PD2 is approximately 20% less than that for PD1, as shown in Figure 8.15.

8.9 PDN vs. PD Implementations

The total run times and the individual component times (averaged over all problems P1 through P4) for the PDN and PD implementations are compared in Figure 8.22. The PDN implementations outperform the PD implementations in every respect. The number of iterations, initialization time, RP solution time and dual time are all less for the PDN implementations than for their PD counterparts.

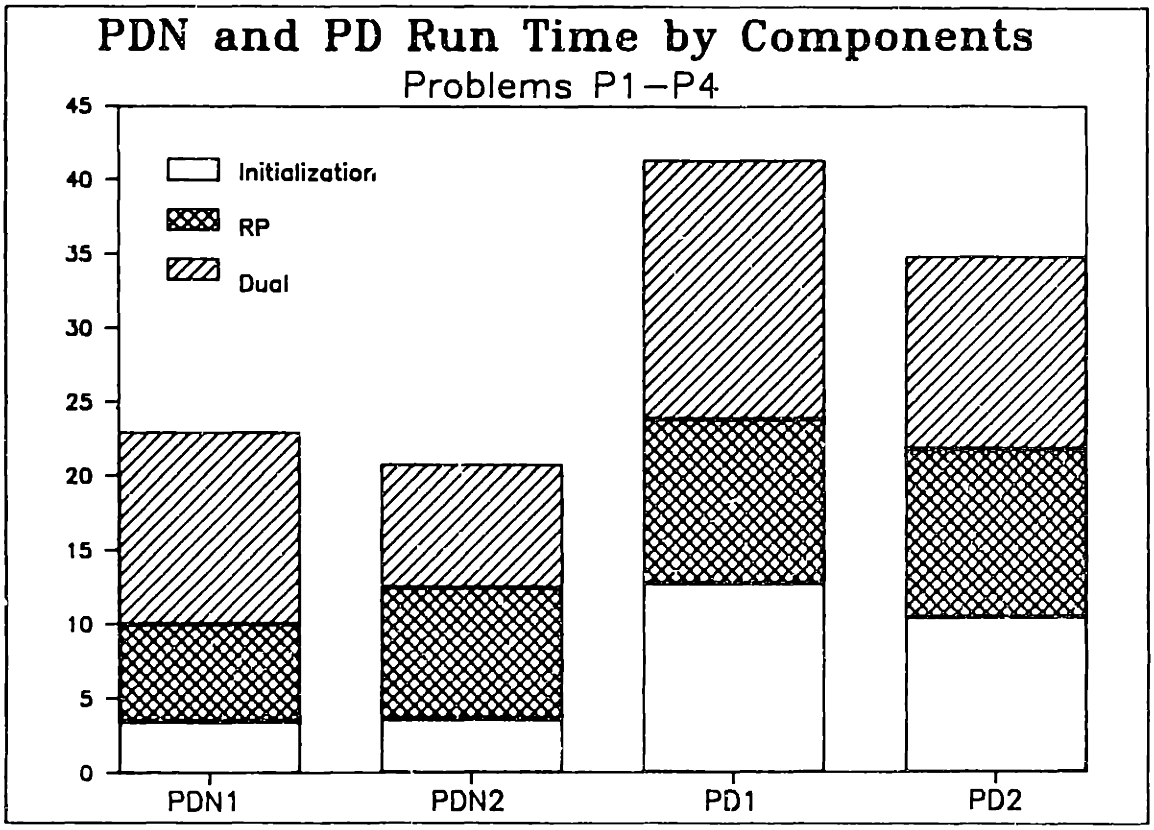

Figure 8.22: PDN and PD Run Time by Components

8.9.1 PDN vs. PD: Number of Iterations

On average, the PD implementations perform 13% more iterations than do the PDN implementations. To explain this result, first recall from Chapter 7 that an iteration refers to the number of dual adjustment steps performed. Hence, the PD implementations perform 13% more dual adjustment steps than do the PDN implementations. Figure 8.23 compares the sequence of dual objective function values generated by the PDN and PD implementations for problem P3.2. The PD implementations, in general, either achieve relatively substantial ascent of the

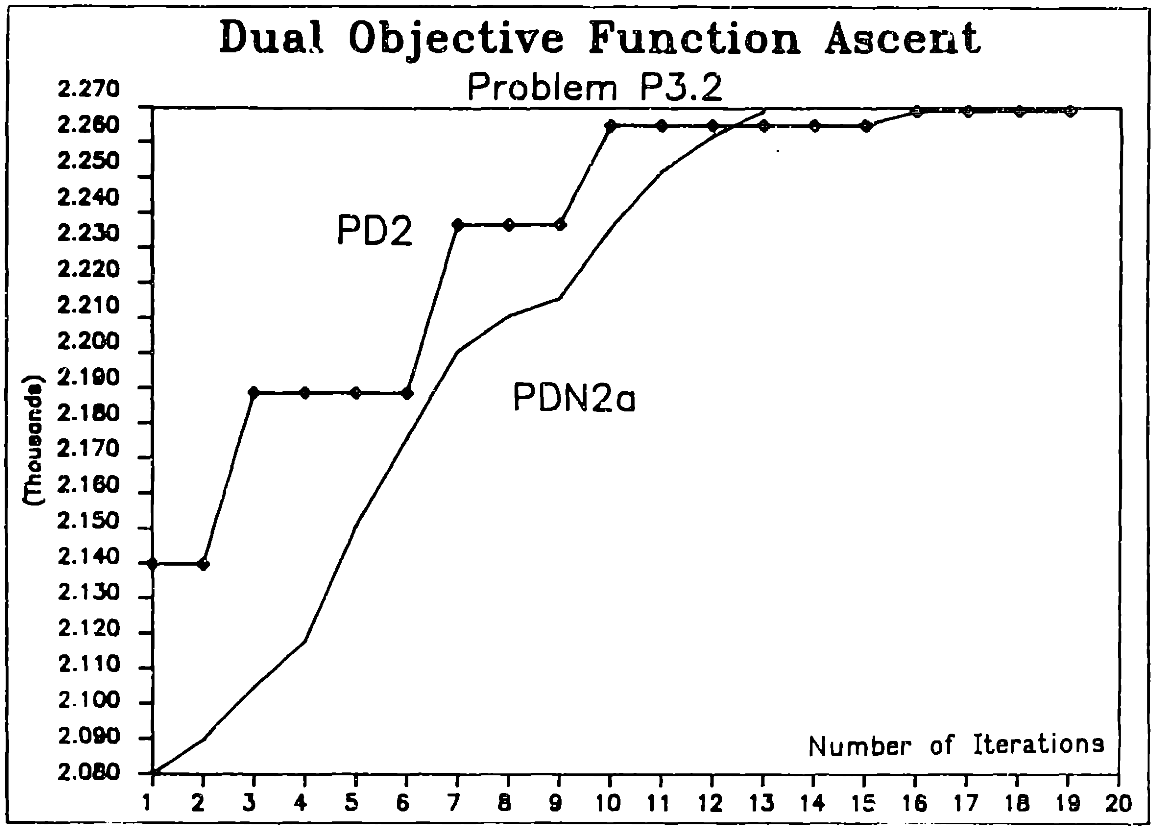

Figure 8.23: PDN vs. PD: Dual Objective Function Ascent

objective function or achieve no ascent at all. In those steps where ascent is not achieved, an alternative shortest path is located which is not yet included in the simplex tableau. In the PDN implementations, although relatively little ascent per iteration is achieved, each iteration does achieve strict ascent. (Theorem 6.1 shows that the PDN algorithm achieves strict ascent at each iteration because the network based nature of the algorithm permits the consideration of all shortest paths simultaneously. Hence, the "next" shortest path not included in RN is always strictly longer and strict ascent is achieved with each dual price adjustment.)

As Figure 8.23 shows, if it were not for these "no-ascent" iterations, the PD algorithm would achieve the optimal solution in fewer iterations than the PDN algorithm. However, since the PD algorithm generates paths on an "as-needed" basis, it gets "stuck" at the same dual objective function value for repeated iterations. Meanwhile, the PDN algorithm continues its "slow but sure" ascent. The end result being that PDN achieves optimality in fewer dual adjustment steps (i.e. iterations) than PD.

Although on average the PD implementations perform fewer iterations than the PDN implementations, the PDN implementations require more iterations in problems P4 than the PD implementations. Figure 8.24 shows that as total demand increases in problems P4, the number of required iterations grows more quickly for PDN than for PD. This difference stems from the inconsistent definition of an iteration. One

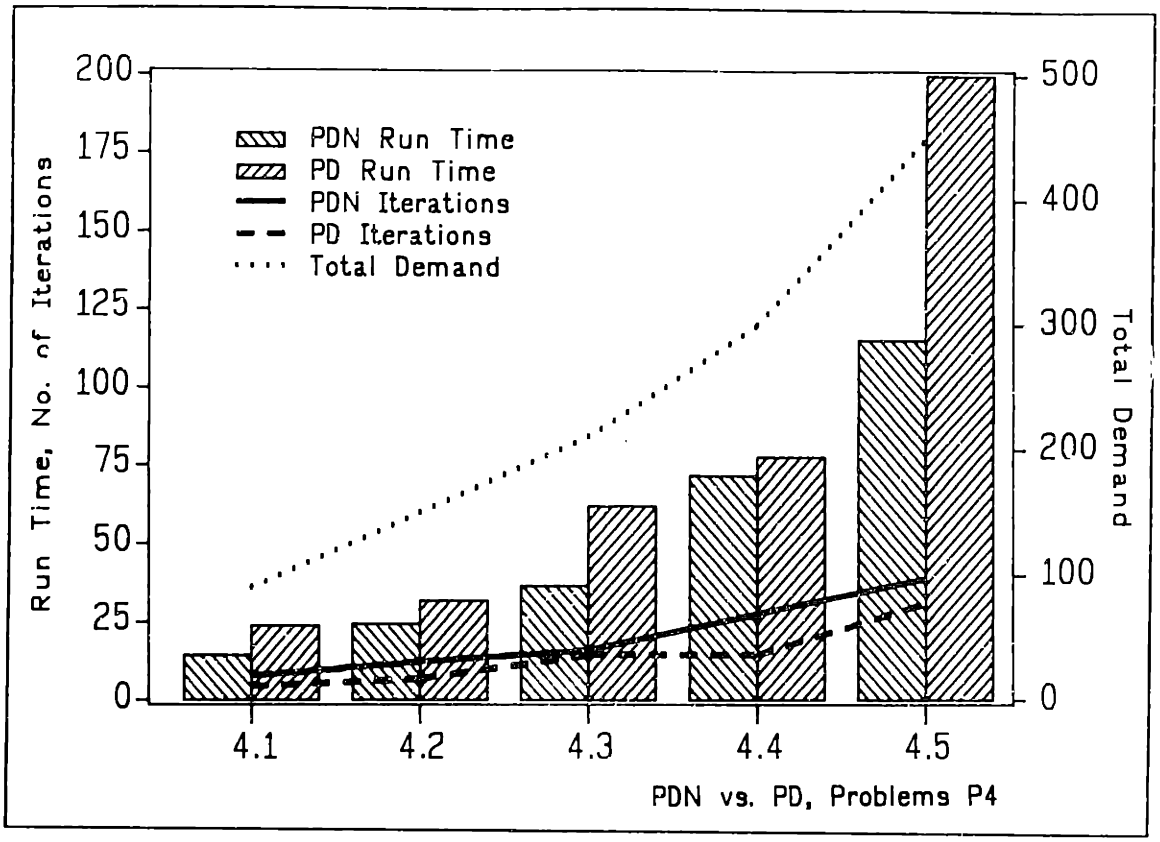

Figure 8.24: PDN vs. PD: Run Time, Number of Iterations and Total Demand

iteration of the PD method corresponds to the solution of one RP sub-problem. However, as explained in Chapter 7, possibly several iterations of the PDN algorithm correspond to the solution of one RP sub-problem. As total demand and congestion increase in the network, more RP subproblems must be solved. It is expected that as the number of RP subproblems increases, the number of iterations required in the PDN algorithm will surpass the number required in the PD algorithm. It is important to note however, that although the number of iterations of the PDN algorithm exceeds that of the PD algorithm, the PDN average total run time for problems P4 is about 63% of that for the PD implementations. Averaging over all problems P1 through P4, the PDN implementations require about 55% of the time required by the PD implementations. Hence, as total demand increases, the PDN method, as compared to the PD method, loses some ground due to increases in the number of iterations required to achieve optimality. However, even with increased total demand, the PDN implementations continue to outperform the PD implementations.

To conclude, the PDN implementations solve the test problems, on average, in fewer iterations than the PD implementations because all admissible paths are considered simultaneously and not added one by one as in the PD methods.

8.9.2 PDN vs. PD: Initialization Times

The average initialization times for the PDN implementations are about 3 times less than those for the PD implementations. The sharp contrast in these initialization times is attributable to the relatively large amount of time required to set up the simplex tableaus for the PD implementations. The PDN implementations do not require the set-up of a simplex tableau and thus, within less time, are able to set up the problem as well as determine dual and primal starting solutions.

8.9.3 PDN vs. PD: Dual Times

The average dual time per iteration is about 25% greater for the PD implementations than for the PDN implementations. This is because the PD methods perform more label-correcting proce. .res for each dual price adjustment than do the PDN methods. As previously stated, the PD methods determine θ by running a label-correcting algorithm for each commodity. The PDN methods, however, determine θ by running a label-correcting algorithm for only those commodities involved in the failed flow adjustment step.

8.9.4 PDN vs. PD: RP Solution Times

The average RP solution time per iteration and the total RP solution time is less in the PDN algorithmic implementations than in the PD algorithmic implementations. In fact, repetitions of the Flow Adjustment Algorithm solve the RP subproblems in about 70% of the time it takes the simplex method to solve the RP subproblems. These results should not be further analyzed because both the PD and PDN methods use different RP solution techniques; solve different RP subproblems; and define iterations differently.

8.9.5 PDN vs. PD: Total Run Time

As indicated by Figure 8.22, the PDN implementations solve the test problems in approximately 55% of the time required by the PD implementations. The PDN implementations perform the initialization step; solve the RP problem; and adjust the dual prices faster than do the PD implementations (Table 8.3).

	PDN vs. PD
Total Run Time	<
Average Number of Iterations	<
Average Initialization Time	<
Average RP Solution Time/Iteration	<
Average Dual Time/Iteration	<

Table 8.3: PDN vs. PD

The greatest inefficiency of the PD algorithms is captured in Figure 8.23. The number of dual ascent iterations achieved by the PD implementation for Problem P3.2 is 5, which represents roughly 25% of the total number of iterations performed. The remaining 75% of the iterations do not achieve ascent of the dual objective function because shortest paths are not included in the simplex tableau. If the PD methods were able to indeed include (efficiently) all "necessary" paths of shortest length in the tableau at each iteration, then the number of iterations of the PD algorithms would be significantly reduced. The difficulty, of course, is in determining the "optimal" number or "optimal" selection of paths to incorporate into the simplex tableau at each iteration. This highlights the benefit of the network-based strategy used in the PDN algorithm. It allows the network to be examined on an arc, rather than a path, basis. Consequently, by determining the set of arcs on shortest paths (using a shortest path labeling algorithm), the PDN algorithm can consider all shortest paths without specifically generating them, as potentially required in the PD algorithms.

8.10 DW vs. PDN and PD Implementations

The average total run times and the individual component times (averaged over all problems P1 through P4) can be compared for the DW, PDN and PD solution methodologies. As shown in Figure 8.25, the DW algorithmic implementation outperforms all other implementations. Averaging over all problems, the number of iterations required to achieve optimality; the total RP solution time; the total dual time; and consequently, the total run time are less for the DW implementation than for any other implementation. Initialization time for the DW method is less than that for the PD methods but greater than that for the PDN methods.

8.10.1 DW vs. PDN and PD: Number of Iterations

In the Dantzig-Wolfe and primal-dual methods, additional columns or paths are generated as necessary at each iteration. It stands to reason that the method which generates the most (good) paths at each iteration is most likely to solve the problem with the fewest iterations. Figure 8.21 shows that the DW method generates at each iter-

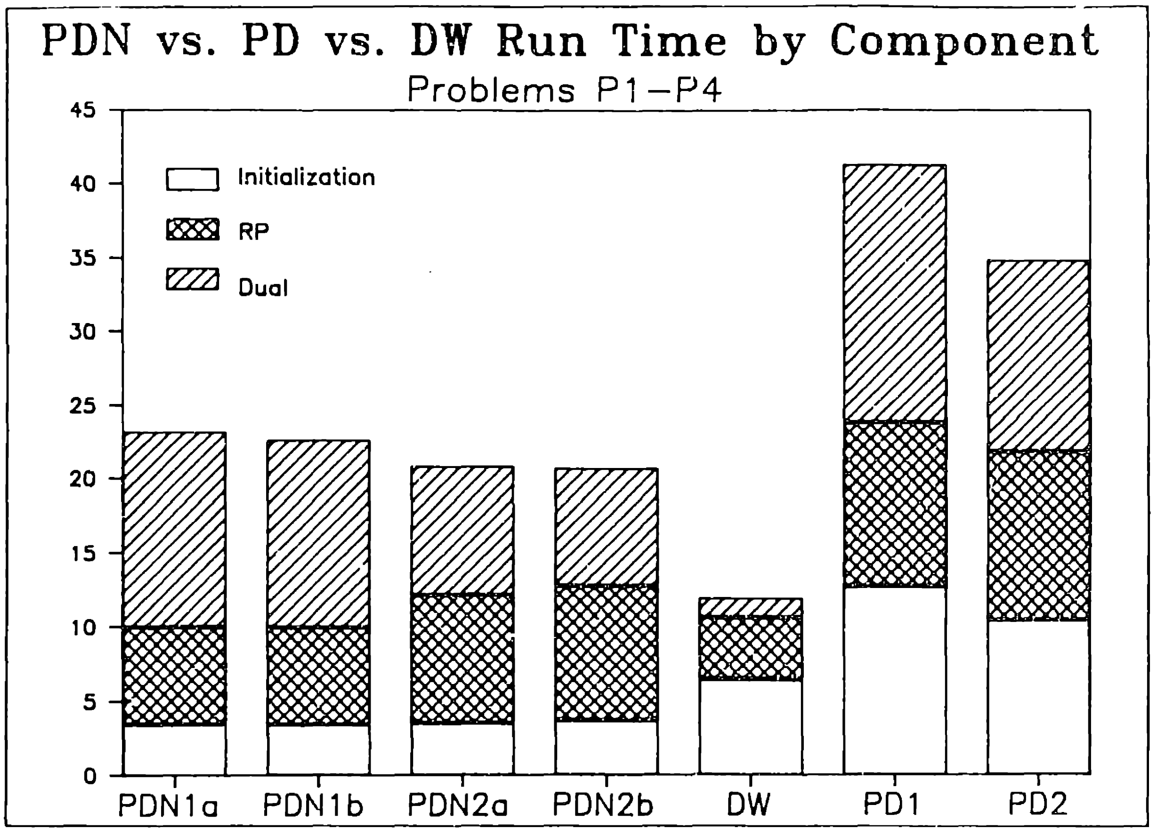

Figure 8.25: PDN vs. PD vs. DW: Run Time by Components

ation, more than twice the average number of columns (paths) generated by the primal-dual methods. This can be explained by the fact that at each iteration the DW method can generate a new path for **each** commodity, whereas the primal-dual based methods can generate only a new path for those commodities determining the optimal value for θ. Hence, applying the logic above and assuming the generated paths are "good", the DW method should solve the test problems in fewer iterations than the PD method. This deduction is supported by Figure 8.26.

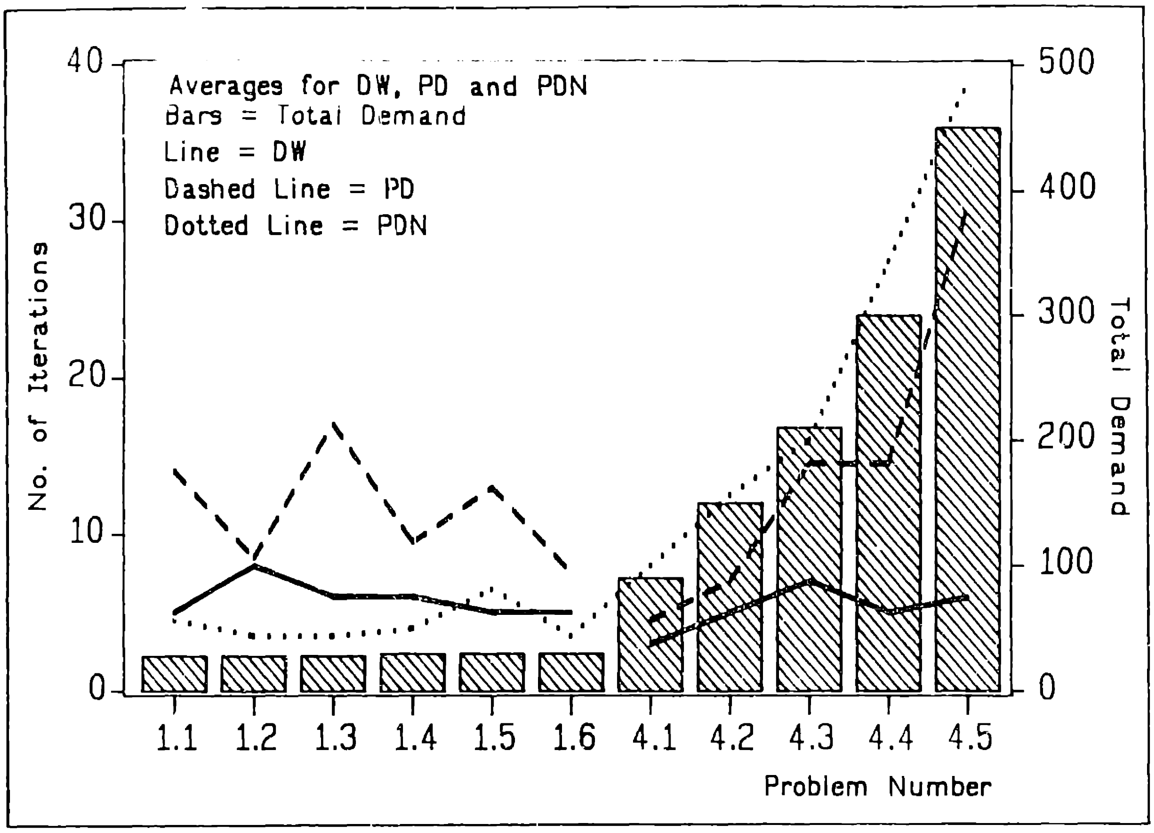

Figure 8.26: PDN, PD & DW: Number of Iterations and Total Demand

Figure 8.26 shows the relationship between total demand and the number of iterations in the PD, PDN and DW solution methods for problems P1 and P4. As the graph indicates, the PD and PDN solution methods are much more sensitive to total demand than are the DW solution methods: the number of iterations grows much more sharply with increases in demand for the PD and PDN methods than for the DW method. For problems P1, with relatively little demand, the PDN solution method requires fewer average iterations than does the DW method. (The PD methods, however, require about twice as many iterations as the DW methods, even for problems P1.) For problems P4, however, where total demand increases by a factor of 5 with problem number, the number of iterations in the DW method is about 1/4 the number in the PDN methods and about 1/3 the number in the PD methods. Averaging over all problems, the number of iterations in the DW method is about 1/2 the number in the primal-dual based methods. The sensitivity of the PD and PDN methods with respect to total demand, and the relative lack of sensitivity of the DW methods, can be explained by considering the sequence of paths generated by the alternative methods at each iteration.

<u>8.10.1.1 DW vs. PDN and PD: Path Generation</u>

As the total demand increases, the congestion level in the network increases and consequently, the competition among commodities for capacity increases. This means that unlike the uncongested case, flow will not simply be assigned to its one shortest path. Instead, as congestion grows, it is expected that more and more commodities will have to be assigned to longer and longer paths. In the PD and PDN methods, since flow can be assigned only to **shortest** paths, dual prices (for each commodity) must be adjusted so that the lengths of all the paths are at least as great as the longest path to which flow is assigned. Hence, as the number and the lengths of paths used by commodities increase, the expected number of dual price adjustments or equivalently, the expected number of iterations should likewise increase.

Consider now the DW methodology. As before, with an increase in congestion comes an increase in the length of paths utilized by each commodity. The DW method, however, requires only that flow be assigned to the set of shortest paths (i.e. columns) considered at the time of the iteration. Hence, dual prices (for each commodity) are assigned such that the lengths of all the considered paths are at least as great as the longest path to which flow is assigned. Unlike the primal-dual based methods, the DW method does not have to generate all the paths which are shorter than the longest utilized path. Therefore, longer paths will be generated in fewer iterations of the DW method than of the primal-dual methods. To illustrate this, the following example is considered.

Figure 8.27 shows the initial set of shortest paths considered by the PD, PDN and DW methods. The first iteration of the PD and PDN methods assigns all flow to shortest paths. Consequently, arcs (n3,n4) and (n4,n5) become (over-)capacitated by commodity k1 and arc (n8,n9) becomes (over-)capacitated by commodity k2. Using equations [2.32] and [2.33], the PDN and PD algorithms update dual prices as shown in Figure 8.28. Then, the "next" shortest path, denoted p3 (n1-n2-n8-n9-n5-n6), is located. Notice that since path p2 is fully capacitated by commodity k2, path p3 has no residual capacity and thus, will not help advance the solution towards primal feasibility.

Examine now, the path generation processes of the DW implementation. Applying the DW method to the network in Figure 8.27 results in

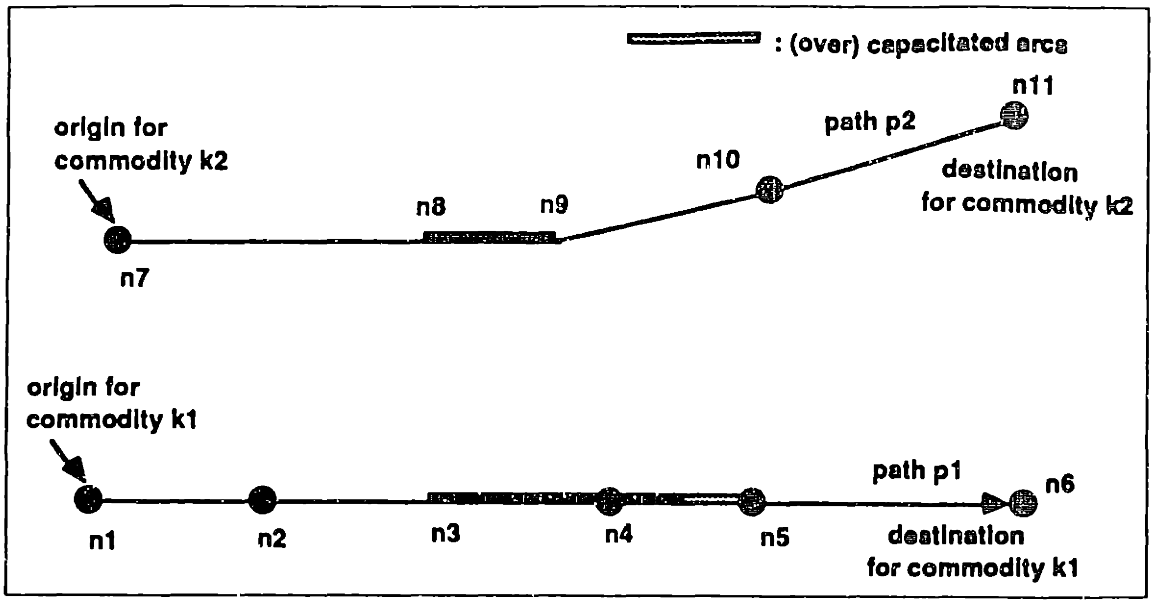

Figure 8.27: Initial Set of Shortest Paths

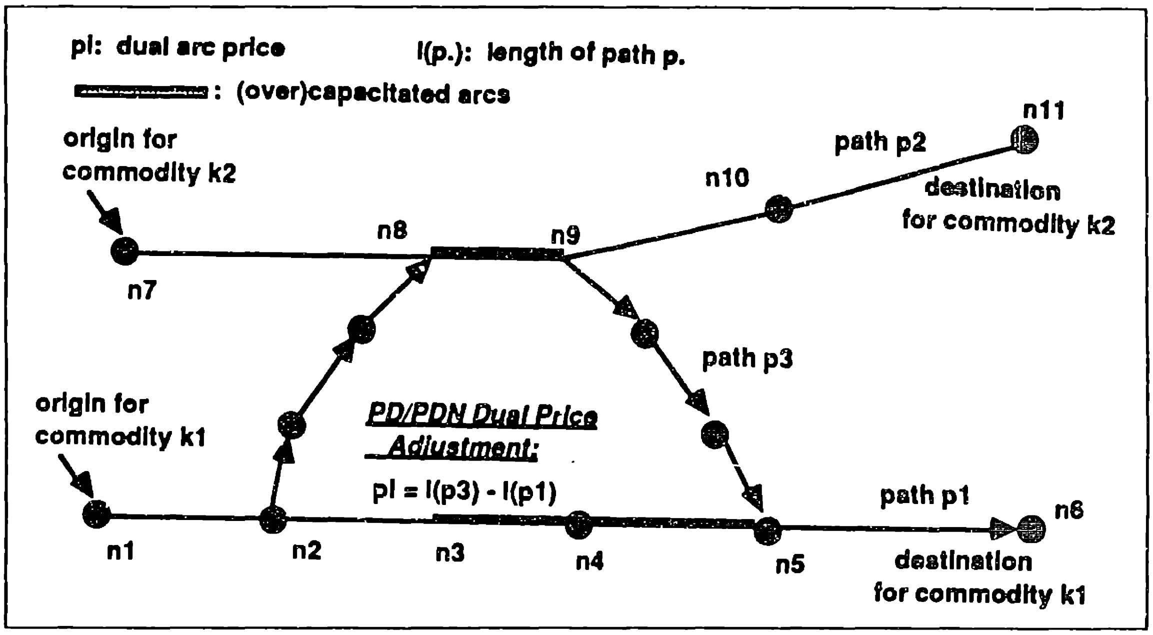

Figure 8.28: PD and PDN Solution Methodologies

the assignment of the dual prices and the generation of the new paths, denoted p4 (n1-n2-n5-n6) and p5 (n7-n8-n10-n11), shown in Figure 8.29. Paths p4 and p5 are longer than path p3 buc unlike p3, both p4 and p5 have residual capacity and both have total length less than the artificial arc. Thus, paths p4 and p5 can be used to advance the primal solution closer to optimality.

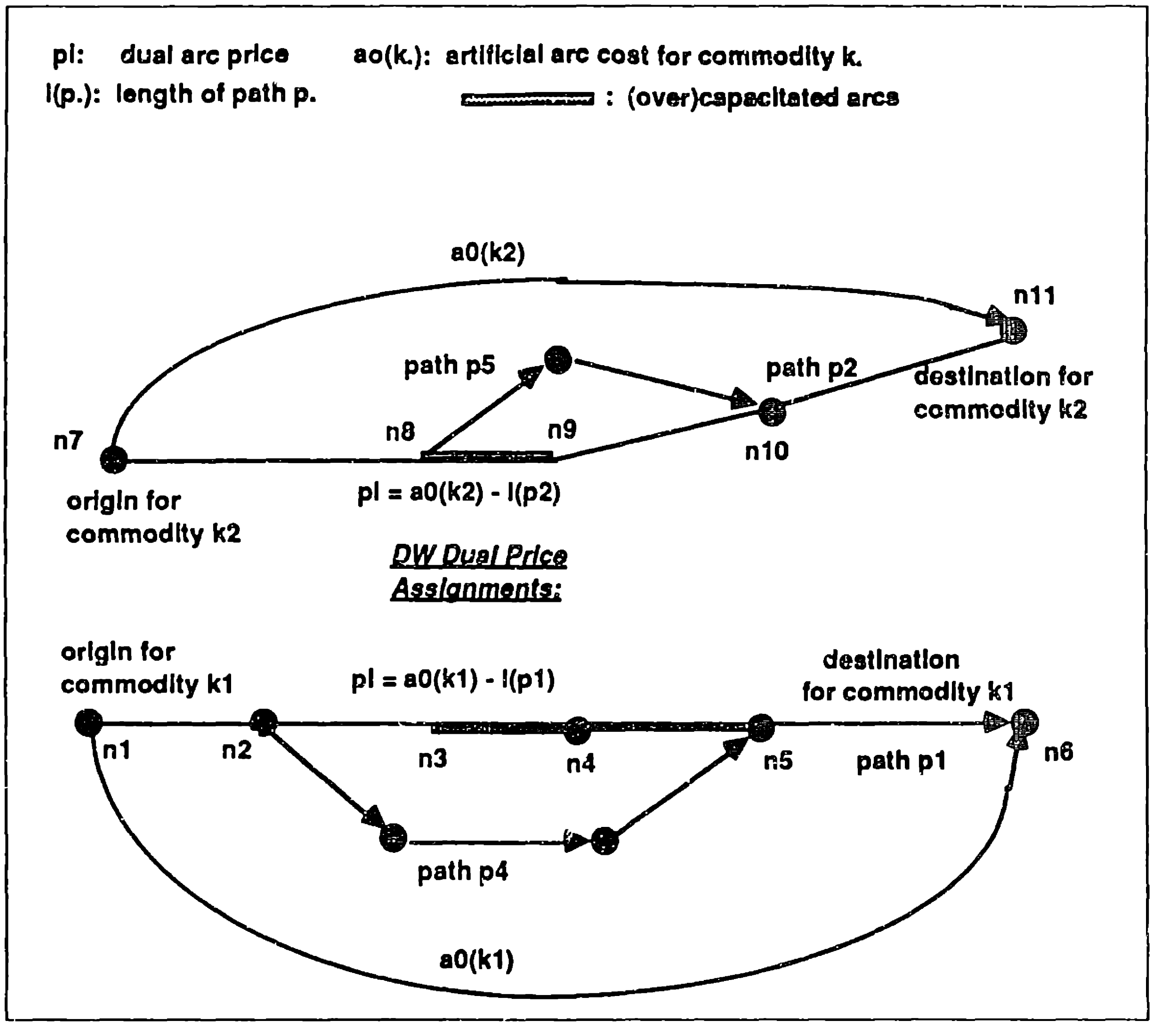

Figure 8.29: DW Solution Methodology

This example provides insight into the differences between the path generation processes of the Dantzig-Wolfe and primal-dual solution methodologies. The Dantzig-Wolfe method is less likely than the primal-dual based methods to generate "capacitated" paths. This is because in solving the RP problem, the Dantzig-Wolfe method determines actual dual prices, unlike the primal-dual methods which determine directions of change in dual prices. For the Dantzig-Wolfe method, the dual prices on the capacitated arcs, in general, are large enough so that the capacitated arcs will not be included in the newly generated paths. Thus, the Dantzig-Wolfe method is successful in generating paths with residual capacity which can be then used to hold flow assigned to the artificial arcs. In the primal-dual methods, however, the dual prices on the capacitated arcs represent only directions of change. The actual revised dual prices ensure only that the new path generated is the next shortest path, without regard to residual capacity. This shortest path generation scheme is not very successful in satisfying the objective of reducing infeasibilities in the network. The added problem with the shortest path generation scheme is that a path can be generated only for those commodities determining the next shortest path. In the DW method however, one path can be generated for each commodity. Hence, more paths can be generated per iteration in the DW method than in the primal-dual methods.

To conclude, the number of iterations is less for the DW method than for the primal-dual methods because the "uncapacitated" path scheme, as compared to "shortest" path scheme, results in:

i) the generation of more paths; and

ii) the generation of paths which are more effective in achieving the problem objective.

8.10.1.2 DW vs. PDN and PD: Tailing-Off Effect

Dantzig-Wolfe decomposition has been known to exhibit the "tailing-off effect" in solving problems. In other words, as the optimal solution is approached, the improvement in the objective function value achieved at each iteration becomes smaller and smaller. In solving test problems P1 through P4 with the Dantzig-Wolfe decomposition method, Assad [5,6,7] reported that no tailing-off effect occurred. This result is substantiated (as shown for Problems P1, Figure 8.30) when the DW implementation is used to solve these same problems.

8.10.2 DW vs. PDN and PD: Initialization Times

The average initialization time for the PDN implementations is about one-half that for the DW implementation. This results because the DW implementation must, in addition to data input and network generation, set-up the initial simplex tableau.

The average initialization time for the DW implementation, on the other hand, is about 1.5 times less than that for the PD implementations. The increased time for the PD implementations results from the large number of artificial variables and hence, columns which must be generated in the initial tableau.

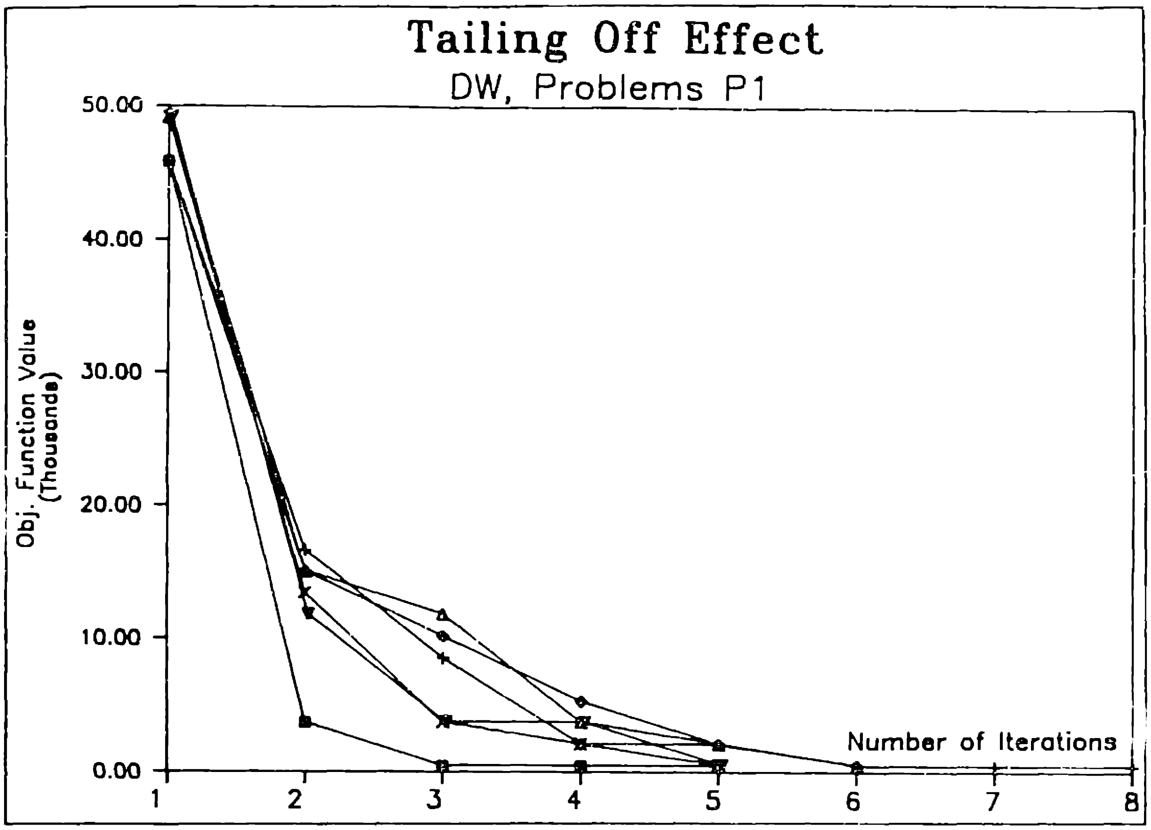

Figure 8.30: Tailing-Off Effect

8.10.3 DW vs. PDN and PD: Dual Times

The total dual time is 6 to 13 times greater for the primal-dual based implementations than for the DW implementation because of differences in:

i) the dual time per iteration required by the two methodologies; and

ii) the number of iterations.

In the DW method, the dual adjustment step involves only the execution of shortest path algorithms for each commodity-- the θ computation is eliminated. As a result, the average dual time per iteration in the DW method takes from 3 to 6 times less than that required in the primal-dual methods. To magnify the difference in total dual times, the number of iterations performed in the DW implementation is about 1/2 the number performed in the primal-dual based implementations.

8.10.4 DW vs. PDN and PD: RP Solution Times

The RP solution times for the primal-dual based methods are roughly 1.5 to 2.5 times longer than those for the DW method. However, the RP time per iteration for the two methods are about the same when averaged over all problems. Thus, the difference in total RP solution times is explained by the increased number of iterations performed by the primal-dual methods as compared to the DW method.

8.10.5 DW vs. PDN and PD: Total Run Time

For the problems with relatively little demand, the PDN solution method requires fewer average iterations than does the DW method. Consequently, as shown in Figure 8.31, the total run time in solving problems P1 is less for the PDN methods than for the DW methods. The PD methods however, require about twice as many iterations as the DW methods, even for problems P1. As a result, for problems P1, the total run time for the PD implementations are about twice that of the DW implementation.

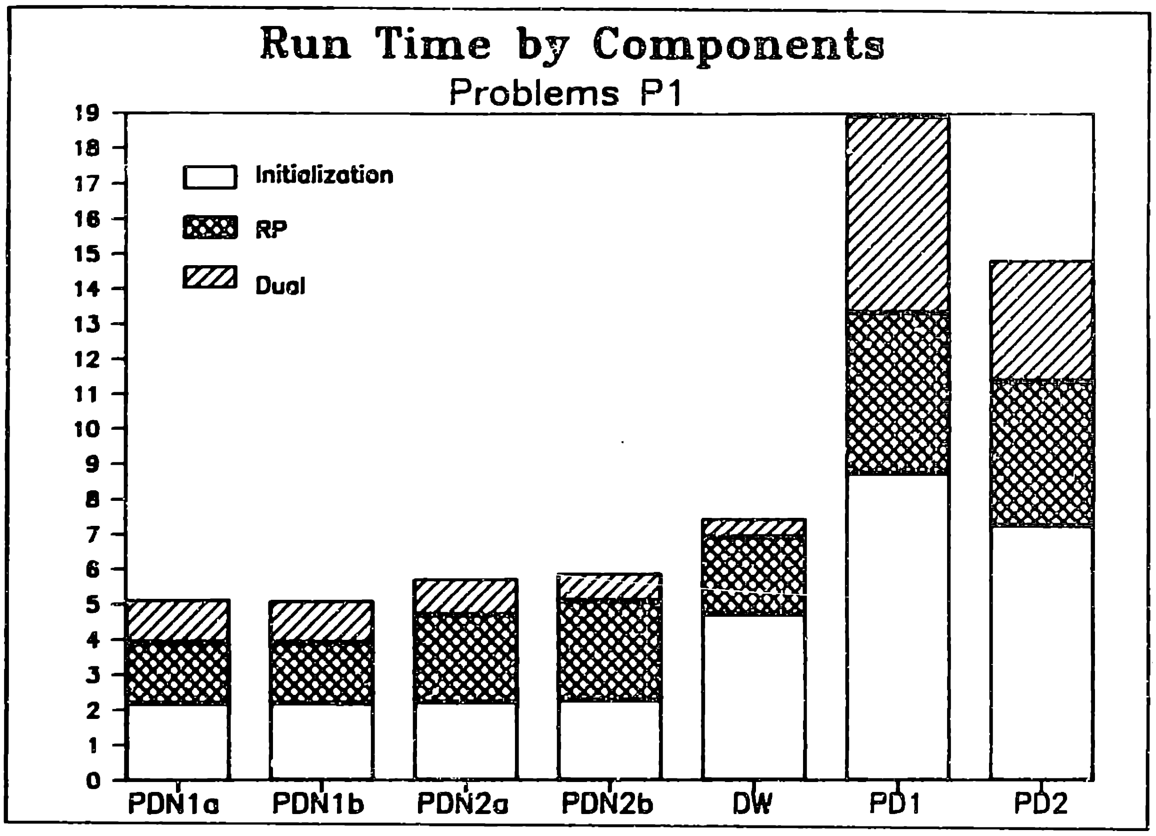

Figure 8.31: Problem P1: Run Time by Components

Contrast the run time results of problems P1 with those for problems P4, shown in Figure 8.32. This time, in the DW method (which is less sensitive to total demand than the primal-dual methods), the number of iterations is about 1/3 the number in the PDN methods and about 1/4 the number in the PD methods. Consequently, for problems P4, the DW run time is about 55% of that for the PDN methods and about 25% of that for the PD methods.

Averaging over all problems, the DW method is about 1.5 to 2 times faster than the PDN implementations and about 3 times faster than the

PD implementations (Figure 8.25).

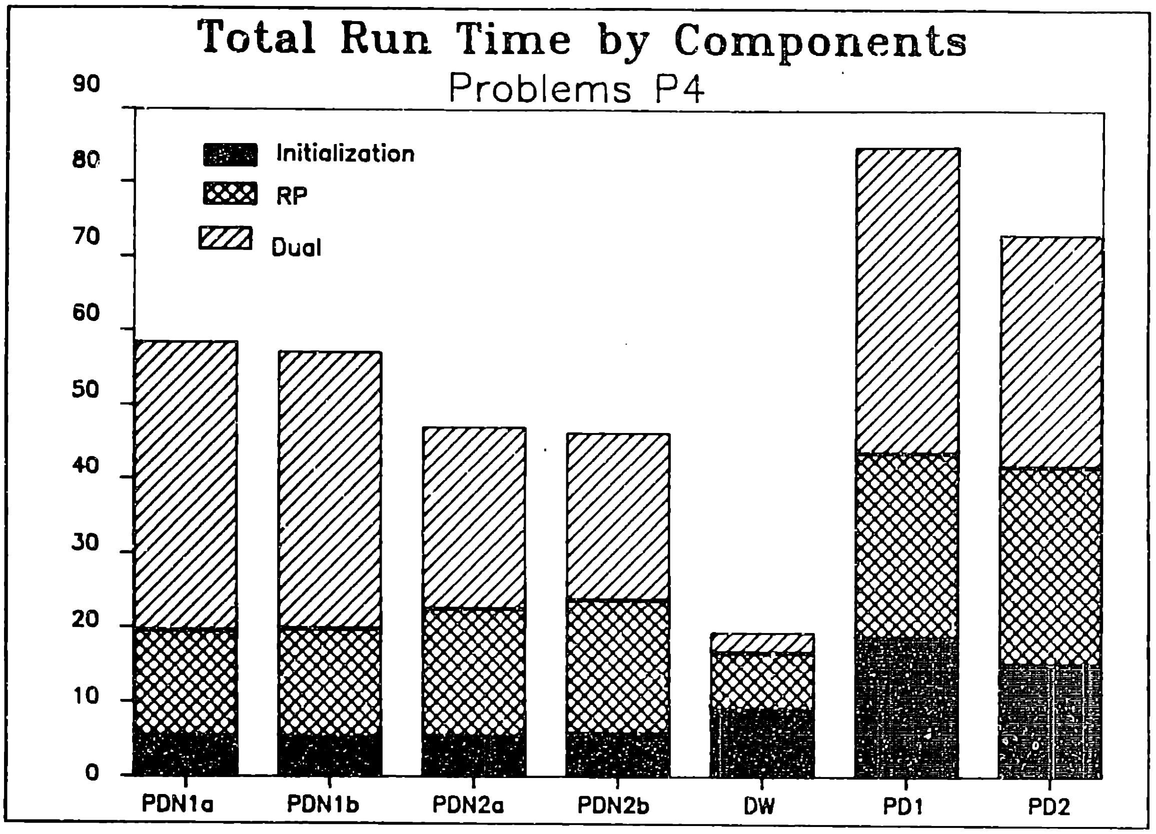

Figure 8.32: Problem P4: Run Time by Components

Table 8.4 summarizes the comparative performance of the DW and primal-dual based algorithmic implementations. On average, the DW implementation solves the test problems faster than the PDN and PD implementations. The success of the DW implementation is attributed to:

i) the efficiency of the DW path generation scheme; and

ii) the reduction in dual time.

	DW vs. PD	vs. PDN
Total Run Time	$<$	$<$
Average Number of Iterations	$<$	$<$
Average Initialization Time	$<$	$>$
Average RP Solution Time/Iteration	$\approx$	$\approx$
Average Dual Time/Iteration	$<$	$<$

Table 8.4: DW vs. PD/PDN

The DW methods, which tend to generate "useful, uncapacitated" paths, require **fewer iterations** than the primal-dual based methods, which instead tend to generate "capacitated, shortest" paths.

In addition, the simplified dual adjustment step of the DW method further increases the run time variation with respect to the PD/PDN methods. The end result is that the DW method is able to solve the CMCF test problems more efficiently than the primal-dual based methods.

8.11 Solution of Large-Scale Problems

As stated in Chapters 1 and 7, the specific application motivating this thesis-- the freight assignment problem, requires the transport of a set of shipments from their origins to their destinations using a fleet of vehicles. In particular, data describing the required pick-up and delivery of less-than-truckload shipments was provided by a large U.S. trucking firm. This problem can be formulated as a linear CMCF problem where:

i) the objective is to minimize total service time for the shipments; and

ii) the underlying network represents the possibilities for shipment service.

For this "real-world" problem (REAL), the nodes of the network are classified into three categories. The first type of node, called a stop node, represents "stops" or locations along a vehicle route. The second type of node, called a source or a sink node, represents shipment origin and destination locations, respectively. An arc joining two stop nodes corresponds to a leg of a vehicle route. The capacity of such an arc equals the vehicle capacity and its cost equals the total time to travel between the locations represented by the from node and the to node of the arc. Arc cost is taken as travel time because the objective of the freight assignment problem is to minimize the total time to service (i.e. load/ transport/ unload) all shipments between their respective origins and destinations. It is assumed that the transportation costs are "sunk" because:

i) the vehicle routes are "fixed"; and

ii) the actual cost incurred is a function only of the number of miles driven and not the size of the shipments transported.

An arc joining shipment k's source node to one of vehicle m's stop nodes represents the possibility of vehicle m picking-up shipment k. Similarly, an arc from one of vehicle m's stop nodes to the sink node of shipment k corresponds to the possibility of vehicle m delivering shipment k. The capacities of these arcs is "infinite" and their costs are equal to zero.

The final node type, called a consolidation node, represents a consolidation terminal at a particular point in time. Consolidation terminals are located throughout the country and serve as "warehouses" where freight can be removed from vehicles, sorted by destination, stored, and transshipped to another vehicle. These terminals help to achieve maximum in-vehicle consolidation. Assignment of a shipment to an arc connecting two consolidation nodes represents "holding" or storage of that shipment at the consolidation terminal for the time period represented by the two nodes. Thus, an arc between two consolidation nodes has cost equal to the time period spanned by the two nodes and capacity equal to the capacity of the consolidation terminal.

The nodes and arcs described above are used to construct the network of service possibilities. For REAL, 2227 shipments must be serviced by this network which contains 1195 nodes and 2820 arcs (of which 1256 are capacitated). Figure 8.33 depicts the way in which a shipment might typically be serviced. Shipment k is picked-up by vehicle m at its origin (node a) and transported through location b before being dropped-off at consolidation center c. Shipment k is then held for two time periods at the consolidation center before it is transferred to vehicle n. Vehicle n transports k through locations d and e before dropping it off at its destination location (node f).

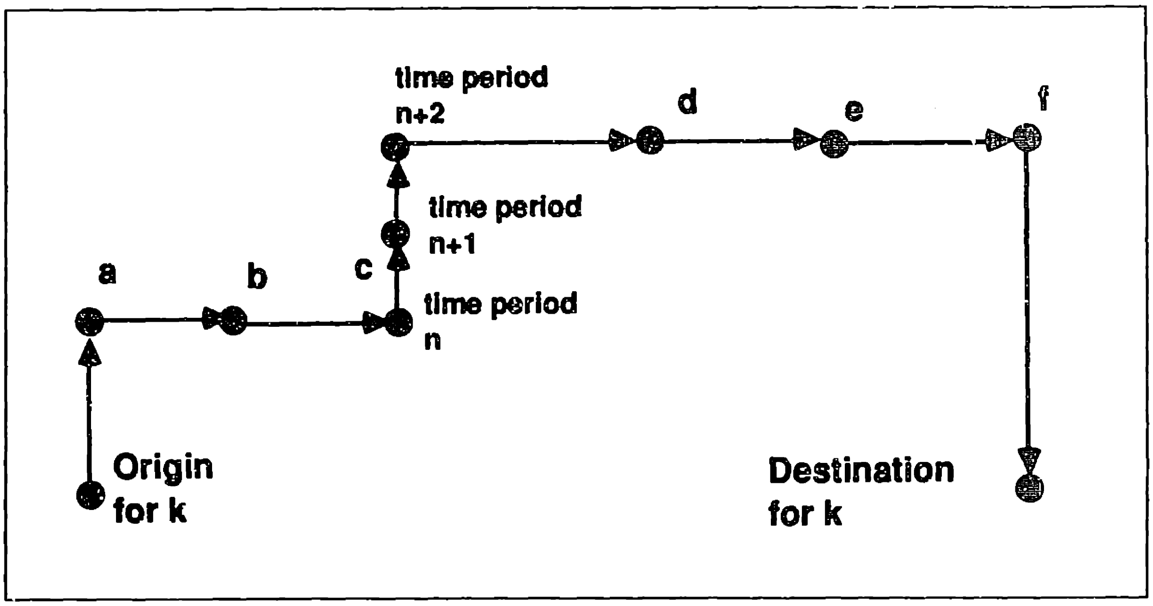

Figure 8.33: Freight Assignment Example

A review of the literature, summarized in Table 1.1, shows that relative to the size of REAL, only small CMCF problems are solved using existing solution techniques. In fact, the largest randomly generated problem reportedly solved (Swoveland [49]) contains up to 356 nodes, 1279 arcs (50 - 60 of which were capacitated) and 6 commodities. Comparatively, REAL has 3.4 times as many nodes, 2.2 times as many arcs, 23 times as many capacitated arcs and 372 times as many commodities.

With respect to real-world size problems, Ali et. al. [3] solves problems whose corresponding LP formulations contain 1395 to 2191 rows and 3165 to 21676 columns. Contrast these LP sizes with that corresponding to REAL. The REAL LP (even using the more efficient tree definition for commodities) contains 61,006 rows and 141,000 columns--approximately 30 times as many rows and 6 to 45 times as many columns as the real-world problems solved by Ali et. al.

In attempting to solve REAL, the simplex-based methods, i.e. LP, DW, and PD, are unable to even begin the solution process. For these solution methodologies, more memory is required to set-up the simplex tableau than is available. Using the DW or PD solution methodologies, the LP for REAL contains 3483 rows and an absolute **minimum** of 4454 columns. Even using a conservative estimate of the number of columns required, this translates to a memory requirement for the DW and PD methods which is approximately 30 times greater than that for the PDN solution methodology. Specifically, to solve REAL, the PDN algorithmic implementations require approximately 11,500 pages (5,888,000 bytes) whereas, the DW solution methodology requires over 320,000 pages (164,398,080 bytes). The difficulty is that the VAXStation II restricts the paging file to at most 20,000 pages and thus, is unable to solve REAL using the DW algorithm. Memory limitations are not exceeded, however, by the PDN algorithm which is able to optimally solve REAL in 117.83 seconds, or approximately 2 minutes on the VAX-Station II.

8.12 Conclusions

The objective of running the LP, PDN, PD and DW algorithmic implementations on the set of test problems (P1 through P4) was not to perform extensive run time analyses but rather, to gain an deeper understanding of the intricate workings of each type of solution methodology and the effect on these methodologies of changes in problem characteristics.

The analysis led to the following generalizations concerning variations in problem data:

i) run time increases as total demand increases;

ii) run time decreases as the average demand size decreases; and

iii) the PDN implementations are more impacted by increases in total demand than are the PD implementations, which are more impacted than the DW implementation.

Variations in the PDN algorithmic implementations showed that essentially no difference results when the flow shift criterion is relaxed to allow a flow adjustment to increase infeasibility on a particular arc. Alterations in the RP formulation, however, do affect algorithmic performance. The most efficient RP formulation, for both the PDN and PD solution methodologies, disallows the creation of OCarcs and assigns all excess flow to artificial origin to destination arcs.

Finally, each algorithmic implementation was evaluated by comparing the number of iterations; initialization time; RP solution time;

dual time; and total run time. For problems P1 through P4, an overall ranking of the solution methodologies, based on these performance measures, is as follows:

1) DW methodology;

2) PDN methodology;

3) PD methodology; and finally,

4) LP methodology.

Overall, the DW methodology was the most effective in solving the CMCF problems. Its effectiveness is attributed to the fact that it requires fewer iterations than either the PDN or PD methodologies. Fewer iterations are required in the DW implementation, as compared to the PDN and PD implementations, because:

i) more columns (paths) are generated per iteration; and

ii) relatively more "uncapacitated" paths are generated.

The more effective path generation scheme of the DW implementation results because, unlike the primal-dual based methods, the DW method does not guarantee the satisfaction of the complementary slackness condition (equation [2.12]) requiring that all flow be assigned to shortest paths at all times.

Attaining second rank is the PDN methodology. Its superior performance over the PD method is primarily attributed to the fact that the network-based nature of the PDN algorithm, as opposed to the column generation-based nature of the PD algorithm, allows the consideration of all shortest paths without explicit generation. In addition to this benefit, in the PDN algorithm, the initialization time, the RP time per iteration and the dual time per iteration are all less than in the PD algorithm. This is explained, at least partially, by the fact that at each iteration, the PDN algorithm can restrict its consideration to a subset of commodities, whereas the PD algorithm considers all commodities.

Finishing last, the LP solution methodology is unable to compete with any of the other solution methodologies, particularly as the number of commodities, nodes and/or arcs in the problem increase. The downfall of the LP solution methodology is that it uses no decomposition and hence, the size of the problem to be solved becomes inordinately large very quickly.

Lastly, only the PDN implementations are able to solve the test problem (REAL) generated from actual data. The LP, PD and DW solution methodologies cannot be used to solve REAL because the simplex algorithm cannot accommodate its large size, even for the DW and PD decomposition methodologies. However, the PDN algorithm, designed specifically to solve large-scale problems, solves REAL to optimality in under 2 minutes on a VAXStation II.

Finally, the in-depth analysis presented in this chapter provides insight which is useful in targeting and evaluating areas for future CMCF research. Some of the ideas and intuition obtained are discussed in Chapter 9.

9. Conclusions

This chapter concludes this thesis by describing in Section 9.1, the major contributions of this research work and in Section 9.2, the proposed directions for future multi-commodity flow research.

9.1 Contributions

Unfortunately the "simple" properties of the **single-commodity** problem do not carry over to the multi-commodity problem and hence, the **multi-commodity flow** problem is particularly difficult to solve. Even so, a review of the literature indicates that for CMCF problems of small to moderate size, rather effective decomposition solution methodologies exist. These decomposition procedures solve the CMCF problem indirectly through an iterative process which uses the simplex algorithm to solve a series of smaller, restricted problems. However, as problem size increases and the size of the restricted problems exceed the limitations of the simplex method, these existing solution techniques are rendered useless. Thus, the research described in this thesis is motivated by the need for a new algorithm to solve large-scale CMCF problems.

Given this motivation, a network-based, primal-dual algorithm (PDN) was developed. The major difference between the PDN algorithm and the conventional primal-dual algorithm (PD) (Chapter 2) is in the solution of the RP and DRP subproblems. The RP/DRP subproblems are solved in the PDN method through repetitions of a network-based Flow Adjustment Algorithm while the PD method uses the simplex algorithm.

The Flow Adjustment Algorithm, by eliminating the size limitations imposed by the simplex algorithm, provides PDN with the flexibility to solve large-scale problems. Besides this primary benefit, the network-based nature of the Flow Adjustment Algorithm affords PDN additional advantages with respect to efficiency.

For example, without explicitly generating every path, the PDN algorithm can assign node labels and simultaneously consider all paths of shortest length. This is a distinct advantage over the column-generation technique of the classical primal-dual algorithm. PD tends to get "stuck" for repeated iterations at the same objective function value because columns (shortest-paths) are generated on an as-needed basis.

Another advantage of the network-based strategy is that the large overhead required in the set-up of the initial simplex tableau is not necessary. As a result, the average initialization times for the PDN algorithms are about 1/3 that of the simplex-based PD algorithms.

Finally, as compared to the simplex-based primal-dual algorithms, the PDN algorithms require less time per iteration to solve the RP problems; update the dual prices; and generate new paths. The time savings is attributed to the fact that the network-based PDN algorithm, unlike the simplex-based PD algorithm, can perform flow shifts and dual price updates for a subset of commodities.

The final result is that even for problems of moderate size (i.e. P1 through P4-- described in Tables 3.2 and 3.3), the PDN algorithm optimally solves the CMCF problems in about 55% of the time required by the PD methods. (Although the proof of optimality for the PDN algorithm relies on the simplex method, optimality was achieved by PDN for all test problems without invoking the simplex algorithm.)

In addition to outperforming the simplex-based primal-dual algorithm, the network-based primal-dual algorithm achieves its primary goal of solving large-scale problems. For the large-scale problem (REAL-- Tables 3.2 and 3.3), the simplex-based methods are unable to accommodate the size of the restricted subproblems. However, the PDN algorithm solves REAL to optimality in under 2 minutes on a VAXStation II.

The contribution of this research work then, is the development of a solution methodology which can be used to solve large-scale CMCF problems which previously could not be solved. Furthermore, even for moderate-size problems (P1 through P4), this new network-based solution methodology is able to solve the CMCF problems in less time than that required by its simplex-based counterpart.

9.2 Future Research

The following sections discuss the following ideas for future multi-commodity flow research:

i) a network-based Dantzig-Wolfe algorithm;

ii) a preprocessing dual-ascent step for the PDN algorithm;

iii) a strategy to relax the conservation of flow constraints rather than the bundle constraints of the CMCF problem; and

iv) a CMCF algorithm using parallel-processing techniques.

9.2.1 Network-Based Dantzig-Wolfe Algorithm

The computational experiments, described in Chapters 7 and 8, show that the Dantzig-Wolfe method solves the test problems (P1 through P4) in less time and with fewer iterations than either the simplex-based or the network-based primal-dual algorithms. The increased efficiency of the Dantzig-Wolfe algorithm is attributed primarily to two points. First, the Dantzig-Wolfe algorithm requires fewer iterations because:

i) it is able to generate more paths at each iteration; and

ii) the paths which are generated tend to be more effective in minimizing total cost as compared to the shortest paths generated by the primal-dual method.

Second, the Dantzig-Wolfe method has the added advantage that only a shortest path algorithm is required to update dual prices and generate new paths. The elimination of the dual adjustment size (i.e. θ) computation results in substantial dual time savings for the Dantzig-Wolfe algorithms over the primal-dual algorithms.

Given that the Dantzig-Wolfe algorithm outperforms the primal-dual algorithms, the question arises as to whether a network-based implementation of the Dantzig-Wolfe algorithm would be more effective than the network-based primal-dual algorithm. To gain insight into the answer to this question, consider why the primal-dual methodology, and not the Dantzig-Wolfe methodology, was selected as the theoretical basis of the new network-based algorithm.

The primal-dual algorithmic strategy to maintain satisfaction of complementary slackness conditions alters the formulation of the RP subproblem by:

i) permitting the elimination of actual arc costs; and

ii) simplifying the objective to that of finding a feasible flow assignment.

The real benefit of this simplified RP problem is that its corresponding DRP problem is also simplified. In fact, the optimal dual prices for the DRP problem can be immediately determined for all arcs with flow strictly less than or strictly greater than capacity. Furthermore, for all arcs with flow exactly equal to capacity, the optimal value of the dual arc price is bounded between 0 or -1 and +1.

By comparison, consider the RP and DRP problems associated with the Dantzig-Wolfe methodology. The Dantzig-Wolfe RP subproblems require the solution, over a restricted network, of a minimum-cost CMCF problem. The optimal dual arc prices for the corresponding DRP problem, unlike the DRP problem for the primal-dual method, are bounded only by non-negativity requirements. Furthermore, the only restriction placed on the dual arc prices is that their value be equal to zero if total arc flow does not equal capacity. This does not mean, as in the primal-dual method, that the dual prices can be immediately fixed at the start for all arcs with flow not equal to capacity. Instead, since the solution of the RP problem may (and should, if sufficient demand requires) alter the flow assignment for any arc up to the capacity of that arc, all dual arc prices are subject to change throughout the RP solution process. This problem is not encountered in the primal-dual methodology because the apriori "fixing" of dual arc prices requires only that the flow level on every arc not increase or decrease beyond (not up to as in the Dantzig-Wolfe case) the capacity of the arc, i.e. no arc with flow less than capacity can become over-capacitated and no arc with flow greater than capacity can become under-capacitated.

Consider the implications of the above discussion on the solution of the RP and DRP problems. If the simplex algorithm is used to solve the RP/DRP subproblems, then the differences in the primal-dual and Dantzig-Wolfe RP formulations have no impact. The simplex algorithm is essentially indifferent to solving a LP with objective function costs reflecting true arc costs or one with costs reflecting infeasibilities. However, if the simplex algorithm is not used to solve the CMCF problems, then the primal-dual methodology clearly has an advantage in that the elimination of arc costs; the need to find only a feasible flow assignment rather than an optimal one; the ability to immediately determine the optimal dual arc prices for an entire class of arcs; and the tight bounds placed on the optimal dual arc prices can all prove to be very valuable simplifications in the development of a solution procedure for the restricted subproblems.

The above arguments indicate that the simplifications which eased the development of the network-based primal-dual algorithm are not present for the development of a network-based Dantzig-Wolfe algorithm. In addition, the gains achieved by the network-based primal-dual algorithm, relative to the simplex-based primal-dual algorithm, would not be realized in the Dantzig-Wolfe implementations. Specifically, the PDN algorithm is able to outperform the PD algorithm, in large part, because it is able to consider all shortest paths simultaneously-- without specifically generating each one. A network-based implementation of Dantzig-Wolfe would not, however, have this same advantage. Since the Dantzig-Wolfe algorithm does not explicitly consider shortest paths, node labels cannot be used to define the set of paths to be considered in the restricted network. Hence, unlike the PDN implementation, a network-based implementation of the Dantzig-Wolfe algorithm would not allow the possibility of considering paths which are not yet generated. Lastly, unlike the PDN algorithm, the dual update and path generation step of the Dantzig-Wolfe algorithm is relatively simple and quick. Thus, it would be impossible, even by considering only a subset of all commodities in the network-based implementation, to achieve substantial savings in the dual time.

Thus, the gains achieved by the network-based strategy for the primal-dual method do not materialize for the Dantzig-Wolfe method. Hence, it is **expected** that a network-based version of the Dantzig-Wolfe algorithm, besides being very difficult to implement, would be outperformed by both its simplex-based counterpart and the PDN algorithm.

9.2.2 Preprocessing Dual Ascent Step for the PDN Algorithm

The PDN algorithm begins by setting all dual arc prices to zero and then running shortest path algorithms to determine the commodity-specific dual node prices. If this rather simplistic initialization step were replaced by a more sophisticated initialization process, it is conceivable that the total number of iterations and hence, the run time in the PDN algorithm could be reduced (possibly substantially).

Saviozzi [46] reported some preliminary results using subgradient techniques with a Lagrangean relaxation of the CMCF problem to achieve an advanced starting basis. The idea presented here is to use a pre-processing dual ascent step to determine a "smart" initial set of dual arc prices and then, proceed with the PDN algorithm as previously described. A step by step description of the "simplistic version" of the dual ascent algorithm (which examines one commodity at a time) is as follows:

DO FOR EACH COMMODITY k:

Step 1) Determine the restricted network, RN^k, containing only shortest paths for commodity k;

Step 2) Separate the nodes contained in RN^k into two sets S and S^c-- where set S contains the source node for commodity k and set S^c contains all nodes in RN^k except those in set S.

Step 3) Define the cut set as all arcs in RN^k with from node in set S and to node in set S^c.

Step 4) Test if the cut set capacity (i.e. the sum of the capacities of all arcs in the cut set) is less than the total quantity of commodity k demanded, i.e. b^k. If so, go to Step 5; otherwise go to Step 7.

Step 5) Alter the dual prices on all arcs in the cut set such that the lengths of the shortest paths and the next shortest path for commodity k become equal. (From Result #5, equation [2.30] and the fact that all shortest paths for commodity k are "cut" by the removal from RN^k of the arcs in the cut set, this price adjustment will result in strict ascent of the dual objective function value.)

Step 6) Update RN^k to reflect the addition of the new shortest path(s) and go to Step 3.

Step 7.) For one arc in the cut set, add its to node to set S and remove it from set S^c.

Step 8.) If the destination node for commodity k is included in set S then STOP; else go to step 3.

This preprocessing dual ascent algorithm can be further enhanced by examining more than one commodity (i.e. pairs, several, or all) simultaneously. The expected result is that the simultaneous consideration of commodities will result in more non-zero dual arc prices and hence, further ascent of the dual objective function value. The improved solution, however, will be achieved at the expense of increased complexity and run time. The best strategy can be determined, of course, only through computational experimentation.

9.2.3 Relaxation Strategy and Parallel Computation

A particularly appealing solution strategy for the CMCF would be to relax, rather than the bundle constraints, the conservation of low constraints at each node. This solution approach offers interesting possibilities and, given the single-commodity results of Bertsekas and Tseng [10,11], has the potential of performing well.

Finally, the possibility for parallel computations provides an interesting and possibly fruitful area of multi-commodity flow research. The CMCF problem and the decomposition-based solution approaches lend themselves naturally to parallel computation methods. For example, the commodity-specific shortest path subproblems in the Dantzig-Wolfe method can be simultaneously solved on a parallel computer using a separate processor for each commodity. Similarly, assigning one processor to each commodity in the Primal-Dual methods, results in the simultaneous determination of the optimal dual price adjustment size and likewise, the simultaneous generation of new shortest paths. These parallel solution techniques have the potential of outperforming sequential solution methods by several orders of magnitude.

BIBLIOGRAPHY

[1] Ali, A., and J. Kennington, "LISS: An In-Core Primal Simplex Code for Solving Linear Programs," Technical Report No. 80015, Department of Operations Research, Southern Methodist University (1980).

[2] Ali, A., R. Helgason and J. Kennington, "An Air Force Logistics Decision Support System Using Multicommodity Network Models," Technical Report 82-OR-1, Department of Operations Research, Southern Methodist University (1982).

[3] Ali, A., D. Barnett, K. Farhangian, J. Kennington, B. Patty, B. Shetty, B. McCarl, and P. Wong, "Multicommodity Network Problems: Applications and Computations," IIE Transactions, 16, 2, 127-134 (June 1984).

[4] "APEX III Reference Manual," CDC Corp. Publication No. 76070000, Minneapolis, Minn. (1974).

[5] Assad, A.A., "Multicommodity Network Flows-- Computational Experience," Working Paper OR-058-76, Operations Research Center, M.I.T. (October 1976).

[6] Assad, A.A., "Multicommodity Network Flows-- A Survey", Networks, 8, 37-91 (1978).

[7] Assad, A.A., "Solving Linear Multicommodity Flow Problems," Proceedings IEEE International Conference on Circuits and Computers, 1, 157-161 (1980).

[8] Bazaraa, M.S. and J.J. Jarvis, Linear Programming and Network Flows, John Wiley, New York (1977).

[9] Bellmore, M., G. Bennington and S. Lubore, "A Multivehicle Tanker Scheduling Problem," Transportation Science, 5, 36-47 (1971).

[10] Bertsekas, D., "A Unified Framework for Primal-Dual Methods in Minimum Cost Network Flow Problems," Mathematical Programming, 32, 125-145 (1985).

[11] Bertsekas, D. and P. Tseng, "Relaxation Methods for Minimum Cost Ordinary Generalized Network Flow Problems", Operations Research, 36, 1, 93-114 (1988).

[12] Bradley, G.H., G.G. Brown and G.W. Graves, "Design and Implementation of Large Scale Primal Transshipment Algorithms," Management Science, 24, 1, 1-34 (1977).

[13] Bradley, S.P., A.C. Hax, and T.L. Magnanti, Applied Mathematical Programming, Addison-Wesley Publishing Co., Reading, MA (1977).

[14] Carter, E.C. and J.R. Stowers, "Models for Funds Allocation for Urban Highway Systems Capacity Improvements," Highway Research Rec., 20, 84-102 (1963).

[15] Chen, H. and C.G. Dewald, "A Generalized Chain Labelling Algorithm for Solving Multicommodity Flow Problems," Comput. Opns. Res., 1, 437-465 (1974).

[16] Clark, S. and J. Surkis, "An Operations Research Approach to Racial Desegregation of School Systems," Socio-Econ. Plan. Sci., 1, 259-272 (1968).

[17] Cremeans, J.E., R.A. Smith and G.R. Tyndall, "Optimal Multicommodity Network Flows with Resource Allocation," Naval Res. Logist. Quart., 17, 269-280 (1970).

[18] Dantzig, G.B. and P. Wolfe, "Decomposition Principle for Linear Programs," Opns. Res., 8, 101-111 (1960).

[19] Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, Princeton, NH (1963).

[20] Ford, L.R. and D.R. Fulkerson, "A Suggested Computation for Maximal Multicommodity Network Flows," Management Sci., 5, 97-101 (1958).

[21] Ford, L.R. and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, N.J. (1962).

[22] Geoffrion, A.M., "Primal Resource Directive Approaches for Optimizing Nonlinear Decomposable Systems," Oper. Res., 18, 375-403 (1970).

[23] Gersht, A. and A. Shulman, "A New Algorithm for the Solution of the Minimum Cost Multi-Commodity Flow Problem," Proceedings IEE Conference on Decision and Control," 26, 748-758 (December 1987).

[24] Grigoriadis, M.D. and W.W. White, "A Partitioning Algorithm for the Multicommodity Network Flow Problem," Math. Prog., 3, 2, 157-177, (1972).

[25] Grigoriadis, M.D. and W.W. White, "Computational Experience with a Multicommodity Network Flow Algorithm," Optimization Methods for Resource Allocation, R. Cottle and J. Krarup (Eds.), English Universities Press, 205-225 (1972).

[26] Hartman, J.K. and L.S. Lasdon, "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems," Networks, 1, 333-354 (1972).

[27] Held, M., P. Wolfe, and H. Crowder, "Validation of Subgradient Optimization," Math. Programming, 6, 62-88 (1974).

[28] Ho, J.K. and E. Loute, "Computational Experience with Advanced Implementation of Decomposition Algorithms for Linear Programming," Mathematical Programming, 27, 283-290 (1983).

[29] IBM, IBM Mathematical Programming System Extended/370 (MPSX/370) Program Reference Manual, SH19-1095-3 (December 1979).

[30] Jarvis, J.J. and P.D. Keith, "Multi-Commodity Flows with Upper and Lower Bounds," Working Paper, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta (1974).

[31] Jewell, W.S., "Multi-Commodity Network Solutions," ORC 66-23, Operations Research Center, University of California, Berkeley (1966).

[32] Jewell, W.S., "A Primal-Dual Multi-Commodity Flow Algorithm," ORC 66-24, Operations Research Center, University of California, Berkeley (1966).

[33] Jorgenson, N.O., "Some Aspects of the Urban Traffic Assignment Problem," Graduate Report, Institute of Transportation and Traffic Engineering, University of California, Berkeley (1963).

[34] Kennington, J.L., "Solving Multicommodity Transportation Problems Using a Primal Partitioning Simplex Technique," Technical Report CP 75013, Department of Computer Science and Operations Research, Southern Methodist University (1975).

[35] Kennington, J.L. and M. Shalaby, "An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems," Technical Report IEOR 750010, Department of Industrial Engineering and Operations Research, Southern Methodist University (1976).

[36] Kennington, J.L., "A Survey of Linear Cost Multicommodity Network Flows," Operations Research, 26, 2, 209-236 (1978).

[37] Lasdon, L.S., Optimization Theory for Large Systems, MacMillan Co., New York (1970).

[38] Maier, S.F., "A Compact Inverse Scheme Applied to a Multicommodity Network with Resource Constraints," Optimization Methods for Resource Allocation, Richard Cottle and Jacob Krarup (Eds), The English Universities Press, London, 179-203 (1974).

[39] Marsten, R., "XMP: A Structured Library of Subroutines for Experimental Mathematical Programming," Technical Report No. 351, Department of Management Information Systems, The University of Arizona, Tucson (1979).

[40] Moore, E. "The Shortest Path through a Maze," Proceedings of the International Symposium on the Theory of Switching, Harvard University Press, 285-292 (1957).

[41] Murtagh, B.A. and M.A. Saunders, "MINOS User's Guide," Technical Report 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University (1977).

[42] Papadimitriou, C.H. and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Inc., New Jersey (1982).

[43] Pape, U. "Implementation and Efficiency of Moore Algorithms for the Shortest Route Problem," Mathematical Programming, 7, 2, 212-22 (1974).

[44] Rosen, J.B., "Primal Partition Programming for Black Diagonal Matrices," Numerische Mathematik, 6, 250-260 (1964).

[45] Saigal, R., "Multicommodity Flows in Directed Networks," ORC 67-38, Operations Research Center, University of California, Berkeley (1967).

[46] Saviozzi, G.,"Advanced Start for the Multicommodity Network Flow Problem", Mathematical Programming Study, 26, 221-224, (1986).

[47] Shapiro, J.F., Mathematical Programming: Structures and Algorithms, John Wiley and Sons, Inc., New York (1979).

[48] Sheffi, Y., Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods, Prentice-Hall, Inc., New Jersey (1985).

[49] Swoveland, C., "Decomposition Algorithms for the Multi-Commodity Distribution Problem," Working Paper No. 184, Western Management Science Institute, University of California, Los Angeles (1971).

[50] Swoveland, C., "A Two-Stage Decomposition Algorithm for a Gener-

alized Multi-Commodity Flow Problem," INFOR, 11, 232-244 (1973).

[51] Tang, D.T., "Comments on Feasibility Conditions of Simultaneous Flows in a Network," Opns. Res, 13, 143-146 (1965).

[52] Tomlin, J.A., "Minimum-Cost Multicommodity Network Flows," Opns. Res., 14, 45-51 (1966).

[53] Tomlin, J.A., "Mathematical Programming Models for Traffic Network Problems," Unpublished Doctoral Dissertation, Department of Mathematics, University of Adelaide, Australia (1967).

[54] Wardrop, J.G., "Some Theoretical Aspects of Road Traffic Research," Institute of Civil Engineers, London Proceedings, 1, Part 2, 325-378, June 1952.

[55] Weigel, H.S. and J.E. Cremeans, "The Multicommodity Network Flow Model Revised to Include Vehicle Per time Period and Node Constraints", Naval Res. Log. Quart., 19, 1, 77-89 (1972).

[56] White, W.W. and A.M. Bomberault, "A Network Algorithm for Empty Freight Car Allocation, IBM Systems J., 9, 2, 147-169 (1969).

[57] White, W.W. and E. Wrathall, "A System for Railroad Traffic Scheduling", IBM Scientific Center Technical Report 320-2993, Philadelphia,Pa. (1970).

[58] Wollmer, R.D., "Multicommodity Networks with Resource Constraints: The Generalized Multicommodity Flow Problem," Networks, 1, 245-263 (1972).