

Successive Shortest Path Algorithms For a Class of Network Problems

by

Majed Saleh Al-Ghassab

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

February, 1994

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600



Order Number 1360431

Successive shortest path algorithms for a class of network problems

Al-Ghassab, Majed Saleh, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106



**SUCCESSIVE SHORTEST PATH ALGORITHMS
FOR A CLASS OF NETWORK PROBLEMS**

BY

MAJED SALEH AL-GHASSAB

A Thesis Presented to the
FACULTY OF THE COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

THE LIBRARY
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN - 31261, SAUDI ARABIA

FEBRUARY, 1994

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

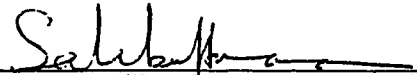
This thesis, written by

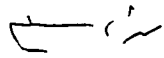
Majed Saleh Al-Ghassab

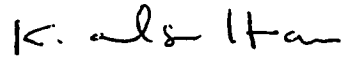
under the direction of his thesis committee, and approved by all the members,
has been presented to and accepted by the Dean, College of Graduate Studies, in
partial fulfillment of the requirement for the degree of

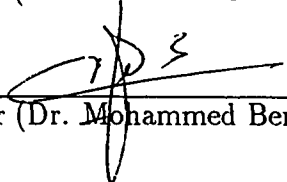
MASTER OF SCIENCE IN SYSTEMS ENGINEERING


Thesis Committee

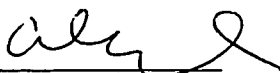

Chairman (Dr. Salih O. Duffuaa)


Member (Dr. Shokri Selim)


Member (Dr. Khalid Al-Sultan)


Member (Dr. Mohammed Ben Daya)


Dr. Khalid Al-Sultan
Department Chairman


Dr Ala H. Al-Rabeh
Dean, College of Graduate Studies

Date: 1/3/94



This thesis is dedicated to

my parents

Acknowledgements

All gratitudes are due to Allah, the Lord of the worlds, May peace and blessings be upon Mohammed the last of the messengers.

Acknowledgement is due to King Fahd University of Petroleum and Minerals for the generous help and support for this research.

I would like to express my deep appreciation to my advisor Dr. S.O. Duffuaa, Associate Professor of Systems Engineering, for his encouragement, guidance, help and suggestion in carrying out this work.

Also, I would like to express my deep gratitude to the other committee members: Dr. S. Selim, Dr. M. Ben Daya and Dr. K. Al-Sultan for their consistent support and cooperation.

The assistance of my friends can not go unacknowledged including the hard-work of Mr. Mohammad Ayub Khan, the manuscript type-setter.

Finally, special thanks must be given to my family, especially my brother Hus-sain, for their encouragement and moral support.

TABLE OF CONTENTS

	Page
Dedication	iii
Acknowledgements	iv
List of Tables	ix
List of Figures	xi
Abbreviations	xiii
Abstract (Arabic)	xiv
Abstract (English)	xv
I. Introduction	1
1.1 General Background	1
1.2 Applications	5
1.3 Thesis Objective	6
1.4 Plan of the Proposed Work	7
1.5 Thesis Organization	9
II. Network Terminology and Literature Review	10
2.1 Introduction	10
2.2 Network Terminology and Storage Scheme	11
2.3 Literature Review	14
2.3.1 Shortest Path Problem	14
2.3.2 Assignment Problem	19
2.3.3 Semi-Assignment Problem	23
2.3.4 Transportation Problem	25
2.4 Conclusion	29

	Page
III. An Efficient Successive Shortest Path Algorithm for the Assignment Problem	30
3.1 Introduction	30
3.2 Description of the SSP Algorithm for the Assignment Problem	32
3.3 The Shortest Path Problem and Labelling Method	42
3.3.1 Labelling Methods	42
3.3.1.1 General Label-Setting Method	43
3.3.1.2 General Label-Correcting Method	44
3.3.2 Implementation Techniques for the Label-Setting Method	45
3.3.3 Implementation Techniques for the Label-Correcting Method	45
3.3.3.1 Pape Method	45
3.4 Computational Aspect and Experimental Design	47
3.4.1 Computational Aspects	47
3.4.2 Experimental Design	50
3.5 Comparison of Dijkstra, Dantzig and Pape Algorithms for for Solving the Shortest Path Problem in the SSP Algorithm	52
3.6 Comparative Computation of Tests	60
3.6.1 Methods for Comparison	60
3.6.2 Results and Analysis of the Comparison	60
3.7 Conclusion	72

	Page
IV. A Successive Shortest Path Algorithm for the Semi-Assignment Problem	73
4.1 Introduction	73
4.2 Description of the SSP Algorithm for the Semi-Assignment Problem	74
4.3 Theoretical Results and Implementation	86
4.3.1 Theoretical Results	86
4.3.2 Implementation of the SSP Algorithm for the Semi-Assignment Problem	87
4.4 Comparative Computational Tests	89
4.4.1 Experimental Design	89
4.4.2 Computational Comparison	89
4.4.3 Memory Requirements of the Codes	93
4.5 Conclusion	94
V. A Successive Shortest Path Algorithm for the Transportation Problem	95
5.1 Introduction	95
5.2 Description of the SSP Algorithm for the Transportation Problem	97
5.3 Theoretical Results and Implementation	101
5.3.1 Theoretical Results	101
5.3.2 Implementation of the SSP Algorithm for the Transportation Problem	104

	Page
5.4 Comparative Computational Tests	108
5.4.1 Experimental Design	108
5.4.2 Computational Comparison	109
5.4.3 Memory Requirements of the Codes	115
5.5 Conclusion	116
VI. Conclusions	117
6.1 Summary of Results	117
6.2 Conclusions	119
6.3 Future Research	120
References	121
APPENDICES:	126
Appendix A Implementation Techniques for Label-Setting Method	128
Appendix B Listing of SPAN-I Code	139
Appendix C Listing of SPAN-II Code	146
Appendix D Listing of SPAN-III Code	152
Appendix E Listing of SPSN Code	156
Appendix F Listing of SPSAN Code	160
Appendix G Listing of SPTN Code	165
Appendix H Listing of NETGEN Code	170
Vita	173

LIST OF TABLES

Table	Page
3.1	Solution Times in Seconds for 50×50 Assignment Problems .. 53
3.2	Solution Times in Seconds for 100×100 Assignment Problems 54
3.3	Solution Times in Seconds for 200×200 Assignment Problems 55
3.4	Code Specifications for the SSP Algorithm for the Assignment Problem with Different Shortest Path Algorithms 59
3.5	Solution Times in Seconds for 50×50 Assignment Problems with Cost Range 1–100 62
3.6	Solution Times in Seconds for 50×50 Assignment Problems with Cost Range 1–10,000 62
3.7	Solution Times in Seconds for 100×100 Assignment Problems with Cost Range 1–100 63
3.8	Solution Times in Seconds for 100×100 Assignment Problems with Cost Range 1–10,000 63
3.9	Solution Times in Seconds for 150×150 Assignment Problems with Cost Range 1–100 64
3.10	Solution Times in Seconds for 150×150 Assignment Problems with Cost Range 1–10,000 64
3.11	Solution Times in Seconds for 200×200 Assignment Problems with Cost Range 1–100 65

Table	Page
3.12 Solution Times in Seconds for 200×200 Assignment Problems with Cost Range 1–10,000	65
3.13 Code Specifications for the Assignment Problem	71
4.1 Solution Times in Seconds for Semi-Assignment Problems with a Cost Range of 1–1000	91
4.2 Solution Times in Seconds for Semi-Assignment Problems with a Cost Range of 1–10,000	92
4.3 Code Specifications for the Semi-Assignment Problem	93
5.1 Solution Times in Seconds for Transportation Problems with a Cost Range of 1–100	111
5.2 Solution Times in Seconds for Transportation Problems with a Cost Range of 1–10,000	112
5.3 Code Specifications for Solving the Transportation Problem ..	115

LIST OF FIGURES

Figure		Page
1.1	Network Representation	2
1.2	Network Flow Programming Problems Relationship ..	4
2.1	Sorted Forward Star Form	13
3.1	An Example of an Assignment Problem	38
3.2	An Example of $SP(C, A, d)$	40
3.3	50×50 Assignment Problems	56
3.4	100×100 Assignment Problems	57
3.5	200×200 Assignment Problems	58
3.6	50×50 Assignment Problems	66
3.7	100×100 Assignment Problems	67
3.8	150×150 Assignment Problems	68
3.9	200×200 Assignment Problem	69
3.10	100×100 Assignment Problem with Cost Range 1-100 and 1-10,000	70
4.1	An Example of a Semi-Assignment Problem	78
4.2	An Example of $SP(C, X, d)$	81

Figure		Page
5.1	An Example of a Shortest Path ($S(v)$) to an Abundant Node from a Root Node	106
5.2	100 × 100 Transportation Problems with Different Cost Range and Total Supply = 100,000	113
5.3	150 × 150 Transportation Problems with Different Cost Range and Total Supply = 150,000	114
A.1	Label-Setting Iteration	129
A.2	Address Calculation Sort	133

ABBREVIATIONS

AP	alternating path
ASSIGN	code of the Out-of-Kilter algorithm for the assignment problem
CAPNET	code of the specialized primal simplex algorithm
CNR	could not run
CPU	central processing unit
GAP	generalized alternating path
HUNG	code of the Hungarian method
NETGEN	code for generating network flow problems
SAP	shortest augmenting path
SAS	statistical analysis system
SP	shortest path
SPAN-I	code of the SSP algorithm for the assignment problem using Dijkstra shortest path algorithm
SPAN-II	code of the SSP algorithm for the assignment problem using Dantzig shortest path algorithm
SPAN-III	code of the SSP algorithm for the assignment problem using Pape shortest path algorithm
SPSAN	code of the SSP algorithm for the semi-assignment problem
SPSN	code of the SSP algorithm for the semi-assignment problem
SPTN	code of the SSP algorithm for the transportation problem
SUPERK	Out-of-Kilter code
TRANS	code of the Out-of-Kilter algorithm for the transportation problem.

خلاصة الرسالة

اسم الطالب بالكامل : ماجد صالح الغصاب .
عنوان الدراسة : خوارزميات أقصر مسار بالتعاقب لفئة من مسائل الشبكات .
التخصص : هندسة النظم والأساليب .
تاريخ الشهادة : فبراير ١٩٩٤ م .

تركز هذه الأطروحة على فئة من مسائل الشبكات تتمثل في مسائل التخصيص ، شبه التخصيص والنقل . إن الهدف الرئيسي من هذه الأطروحة هو تطوير الخوارزمي المقترح لحل مسائل التخصيص بإستخدام أقصر مسار بالتعاقب لمسائل شبه التخصيص والنقل ، ومن ثم تبرمج هذه الخوارزميات وتقارن مع الخوارزميات الأخرى المتوفرة .

الجزء الأول من هذه الأطروحة مكرس إلى التعرف على أفضل خوارزمي أقصر مسار لحل خوارزمي أقصر مسار بالتعاقب المقترح لحل التخصيص . ولقد تبين أن طريقة وضع الرقعة التي ألفها دايكستر هي أفضل طريقة تتناسب مع طريقة أقصر مسار بالتعاقب . إتضح أن خوارزمي أقصر مسار بالتعاقب لحل مسألة التخصيص هو أفضل خوارزمي فعال بالنسبة لحسابات وقت حل المسائل الغير كثيفة .

لخوارزمي مسائل شبه التخصيص حد حسابي هو $r(3)$ وهو أفضل من كابنت (برنامج لخوارزمية سميلكس المتخصص) وترانز (برنامج خارج كلتر) . لقد تم تطوير وبرمجة خوارزمي أقصر مسار بالتعاقب لمسائل النقل ووجدت أنها أسرع بمعدل ١,٣ مرة من كابنت لمسائل مدى تكلفتها قليل .

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران ، المملكة العربية السعودية
فبراير ١٩٩٤ م

THESIS ABSTRACT

FULL NAME OF STUDENT : MAJED SALEH AL-GHASSAB
TITLE OF STUDY : SUCCESSIVE SHORTEST PATH
ALGORITHMS FOR A CLASS OF
NETWORK PROBLEMS
MAJOR FIELD : SYSTEMS ENGINEERING
DATE OF DEGREE : FEBRUARY, 1994

The focus of this thesis is on a class of network problems consisting of the assignment, the semi-assignment and the transportation problems. The major objective of the thesis is to extend the successive shortest path (SSP) algorithm proposed for the assignment problem to solve the semi-assignment and the transportation problems. Then implement the SSP algorithm for these two problems and test their efficiency in comparison with available algorithms.

The first part of this thesis is devoted to identifying which shortest path algorithm best fits the SSP algorithm proposed for the assignment problem. It has been confirmed that an implementation of a label-setting algorithm by Dijkstra is the best shortest path algorithm which best fits the SSP approach. The SSP algorithm for the assignment problem was found to be the most efficient in terms of computational time for solving sparse problems.

The SSP algorithm for the semi-assignment problem has a computational bound $O(n^3)$ and outperformed CAPNET (an implementation of a specialized simplex algorithm) and TRANS which is an implementation of Out-of-Kilter algorithm. The generalized SSP algorithm for the transportation problem is implemented and found to be 1.3 times faster than CAPNET for low cost range problems.

MASTER OF SCIENCE DEGREE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

Dhahran, Saudi Arabia

February, 1994

CHAPTER 1

INTRODUCTION

1.1 General Background

Network optimization is one of the most important practical branches of mathematical programming. Its importance stems from the wide range of practical problems which can be modeled as networks and the availability of efficient algorithms for solving large network formulations. Efficient algorithms have resulted from the specialization of the simplex method and the special data structures for storage and manipulation of network data. The first algorithm developed for solving the network problem based on the specialization of the simplex method is by Dantzig [22] and Johnson [51]. A simple representation of a network is shown in Figure 1.1.

The structure of the network may also be described as $J \times J$ matrix (see Kennington [3, 53]). Mathematically, the minimal cost network flow problem may be stated as follows:

$$\begin{aligned} &\text{Minimize } CX \\ &\text{subject to} \\ &AX = b \\ &0 \leq X \leq U \end{aligned} \tag{1.1}$$

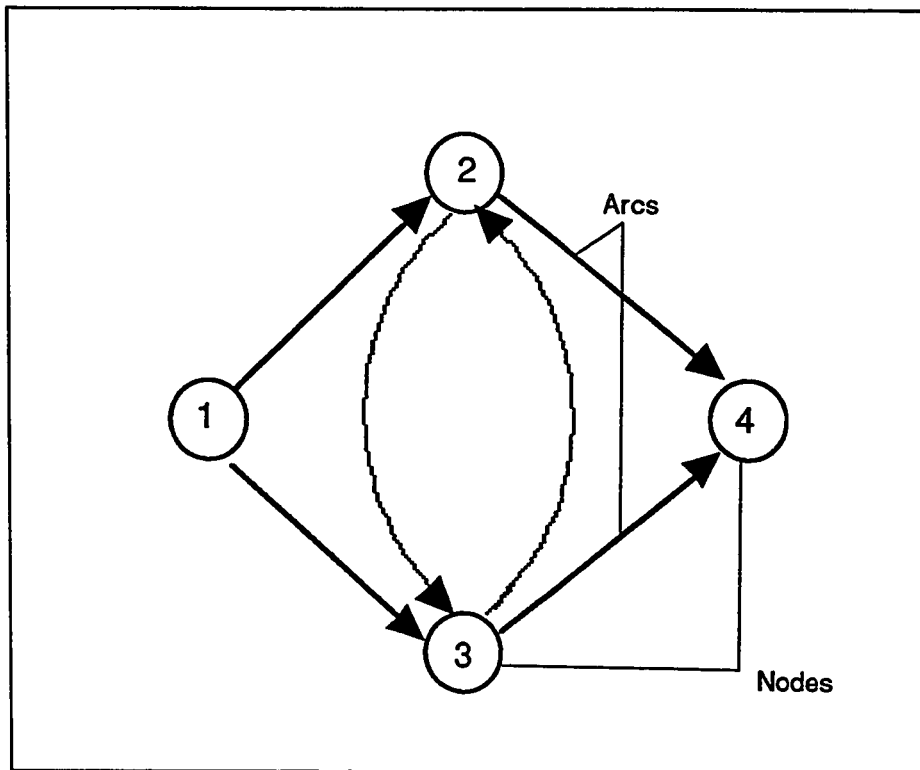


Figure 1.1 Network Representation.

where A is a node-arc incidence matrix, C is a cost matrix, U is a vector of a maximum amount of flow on the arcs, b is a supply/demand vector, and X is the decision variable representing flow on the arcs. Many practically known problems are special cases of the network problem. These are the assignment, semi-assignment, transportation, transshipment, maximal flow, and the shortest path problems. The schematic relationships joining this basic network shown in Figure 1.1 to other network flow programming problems are shown in Figure 1.2. The central point in this figure is the pure, linear, minimum cost flow problem. The problems listed to the left are less general because they are a specialization of the network problem. Problems listed to the right are more general because a network problem is a specialization of each of these problems. The general linear programming problem is also shown in this figure to indicate its relationship to the network programming problems. Algorithms have been developed to solve each of the problem classes in Figure 1.2. Algorithms for the less general problems are more efficient in computation and require less memory than those for the more general problems. The decade of the seventies has experienced tremendous effort in efficient implementation of network algorithms which results in codes 200-300 times faster than general purpose linear programming code [10, 24, 34].

The advantages of using network models are that network representation can represent many real-world systems. Also, it seems that managers accept a network diagram more easily than they do abstract symbols.

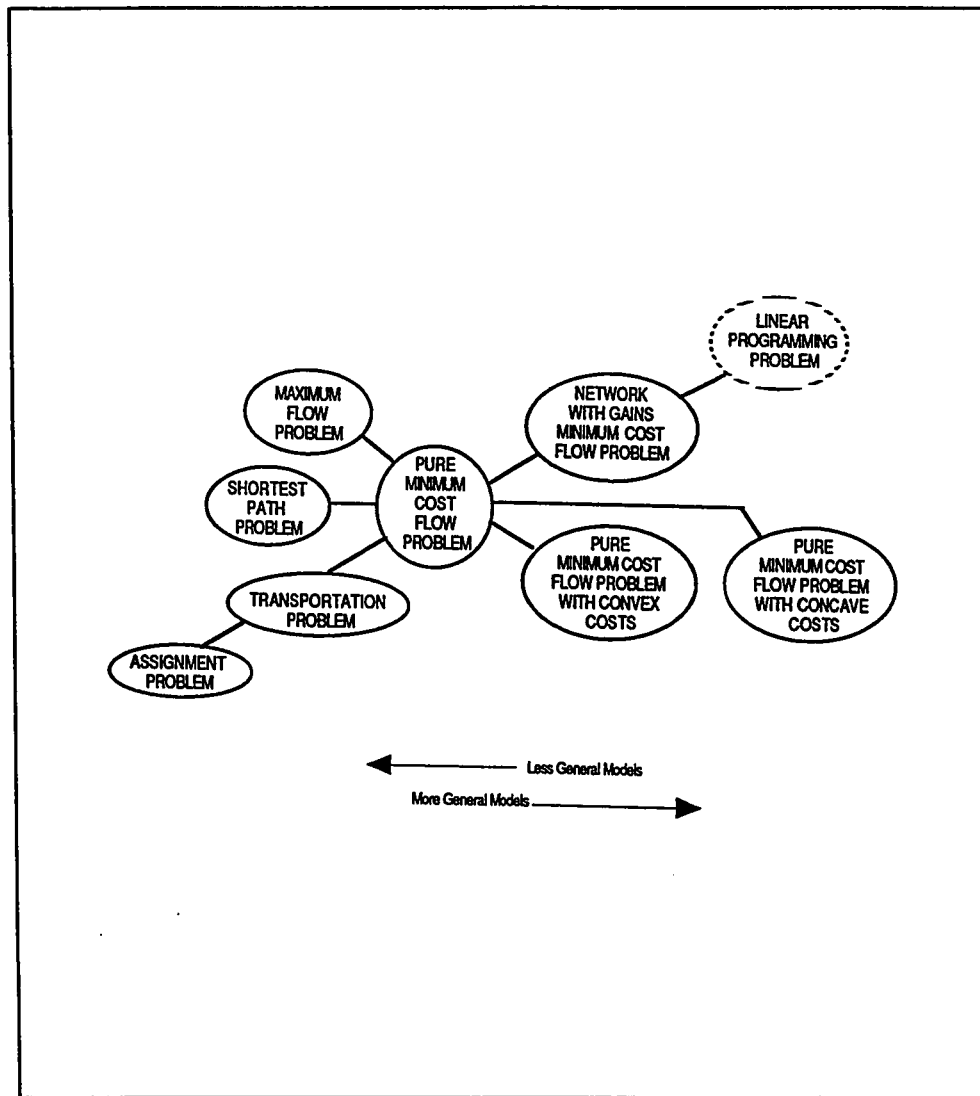


Figure 1.2 Network Flow Programming Problems Relationships [49]

1.2 Applications

As illustrated in Figure 1.1, a network is a collection of nodes and arcs. This representation is useful for modeling a wide range of physical and conceptual situations.

Network flow models and solution techniques provide a rich and powerful framework from which many engineering problems can be formulated and solved. Networks have been used in many applications [38], as inventory systems, river systems, distribution systems, military logistics systems, urban traffic systems, communication systems, facilities location systems, routing and scheduling systems and electrical networks. In fact, network representation is such a valuable visual and conceptual aid to the understanding of relationships between events and objects that it is used in virtually every scientific, social, and economic field.

The focus of this thesis is on algorithmic development of a class of network models. These are, the assignment, semi-assignment and transportation problems. The three models have a wide variety of applications [38].

1.3 Thesis Objectives

In [30] an efficient successive shortest path (SSP) algorithm has been proposed and tested for solving the assignment problem. The primary objective of this work is to extend this approach to semi-assignment and the transportation problems. Therefore, the objectives of the thesis are:

1. To test and identify the most efficient shortest path algorithm (Dijkstra [25], Dantzig [20], and Pape [61]) which fits the SSP framework, for solving the assignment problem.
2. To test and compare the developed SSP for the assignment problem obtained in (1) above with the Hungarian, Out-of-Kilter, and specialized primal simplex algorithms.
3. To extend the SSP algorithm to solve the semi-assignment problem and conduct computational testing with available software.
4. To extend the SSP algorithm to solve the transportation problem and to conduct computational testing with available software.

1.4 Plan of Proposed Work

The proposed research work in the thesis starts with the implementation and the testing of the shortest path algorithms to identify the one that best fits the SSP framework for solving the assignment problem. The following shortest path algorithms are selected for the study: Dijkstra [25], Dantzig [20], and Pape [61]. Empirical testing and comparison is conducted to identify the most efficient one that fits the SSP framework based on solution times and memory requirements for each implemented SSP algorithm.

Then the available codes for solving the assignment problem which are implementations of the Hungarian method, Out-of-Kilter algorithm, and a specialized primal simplex code named CAPNET are compared with the best selected SSP code from the first objective.

The semi-assignment problem is a general case of the assignment problem. In the case of the assignment problem, we have equal number of source and destinations, each source should be assigned to only one destination that minimizes the total cost. However, for the semi-assignment problem we have less number of sources than destinations. Therefore, the extension of the SSP algorithm to solve the semi-assignment problem needs only to consider how to make one source assigned to more than one destination. Furthermore, an implementation of the extended SSP algorithm is compared with the Out-of-Kilter, CAPNET and the

SSP algorithm developed in the first objective that will solve the semi-assignment problem as an assignment problem.

The last part of the thesis focusses on generalizing the SSP approach for the transportation problem. In this case, the concepts developed by Enguist [30] for solving assignment problems have to be modified and theoretical results are developed. Then the modified algorithm has been implemented, tested and compared to CAPNET and an Out-of-Kilter Code.

1.5 Thesis Organization

This thesis is presented in six chapters. In Chapter 2, network definition, terminology and storage schemes, are presented together with the relevant literature.

Chapter 3 presents the SSP algorithm for solving the assignment problem and the methods for solving the shortest path problem. Computational tests of Dijkstra, Dantzig and Pape algorithms for the shortest path problem within the SSP algorithm are reported. Computational procedures and comparisons with the available software are also presented in this chapter.

In Chapter 4, the SSP algorithm for solving the assignment problem is modified to solve the semi-assignment problem. Description, theoretical results and implementation of the modified SSP algorithm is presented. Results and analysis of the comparisons of the modified SSP algorithm with the available software are also shown in this chapter.

Chapter 5 presents the generalization of the SSP algorithm to solve the transportation problem. The theoretical results and the implementation of the generalized SSP algorithm together with computational comparisons of the extended SSP algorithm with the available software for solving the transportation problem are presented. Chapter 6 concludes the thesis.

CHAPTER 2

NETWORK TERMINOLOGY AND LITERATURE REVIEW

2.1 Introduction

Network representation on a graph is easy to understand and imagine how it works. However, mathematically, there is a need to define all the variables associated with the network so that it can be understood and solved without drawing the network. A network can be represented in a computer by many ways; however, we seek the method for representing and storing a network with the minimum memory requirements.

In this chapter, network terminology and storage schemes are presented in Section 2.2. The literature review for a class of network problems, which are the shortest path, assignment, semi-assignments and transportation problems is given in Section 2.3. Section 2.4 concludes this chapter.

2.2 Network Terminology and Storage Scheme

A directed network or simply a network $G(N, A)$ consists of a finite set A of arcs, where each arc $a \in A$ may be denoted as an order pair (u, v) , referring to the fact that the arc is conceived as beginning at a node $u \in N$ and terminating at a different node $v \in N$.

A directed path or a path is a finite sequence of arcs $P = \{a_1, a_2, \dots, a_n\}$ such that for each $i = 2, \dots, n$, arc a_i begins at the end of arc a_{i-1} . P is called a path from node u to node v if a_1 starts at node u and arc a_n terminates at node v . A path P from u to v is called a circuit if $u = v$. A path for which $a_i \neq a_j$ for $i \neq j$ is called arc-simple.

Let $\ell(a)$ or $\ell(u, v)$ denote a nonnegative length associated with arc $a = (u, v)$ of a network. Then we define the length of path P to be $d(P) = \sum_{i=1}^n \ell(a_i)$. Path P from one particular node to another node is called a shortest path if $d(P)$ is the minimum length of any path between these nodes.

A rooted tree, or simply a tree, is a network $T(N_T, A_T)$ together with a node r (the root node), such that each node of N_T , except r , is accessible from r by a unique arc-simple path in T .

In a minimum tree of shortest path tree the unique sequence of nodes beginning with a node $v \neq r$ and leading to r is called the path from v to the root and is

denoted by $S(v)$.

A network may be represented in a computer in several ways [3, 20, 24, 36], and the manner in which it is represented directly affects the performance of network algorithms.

In this thesis, all of the arcs that begin at the same node are stored together and each is represented by recording only its ending node and length. A pointer is then kept for each node which indicates the block of computer memory locations for the arcs beginning at this node. This is motivated by the work done by Dial et al [24]. Dial et al [24] developed one of the fastest algorithms that solves the shortest path problem stores as network with N nodes and A arcs using a linked list structure.

The set of arcs emanating from node u is called the forward star of node u and denoted by $FS(u)$. Similarly, the set of arcs entering a node u is called the reverse star of node u and denoted by $RS(u)$. If the nodes are numbered sequentially from 1 to N and the arcs are stored consecutively in memory such that the arcs in the forward star of node i appear immediately after the arcs in the forward star of node $(i - 1)$, then this method is called, the forward star form, and it requires only $N + 2A$ units of memory. If the arcs of the forward star of each node are ordered by ascending, this is called a sorted forward star form.

Figure 2.1 illustrates the storage of a network in a sorted forward star form. The number in the square is attached to an arc of the network diagram is the arc length.

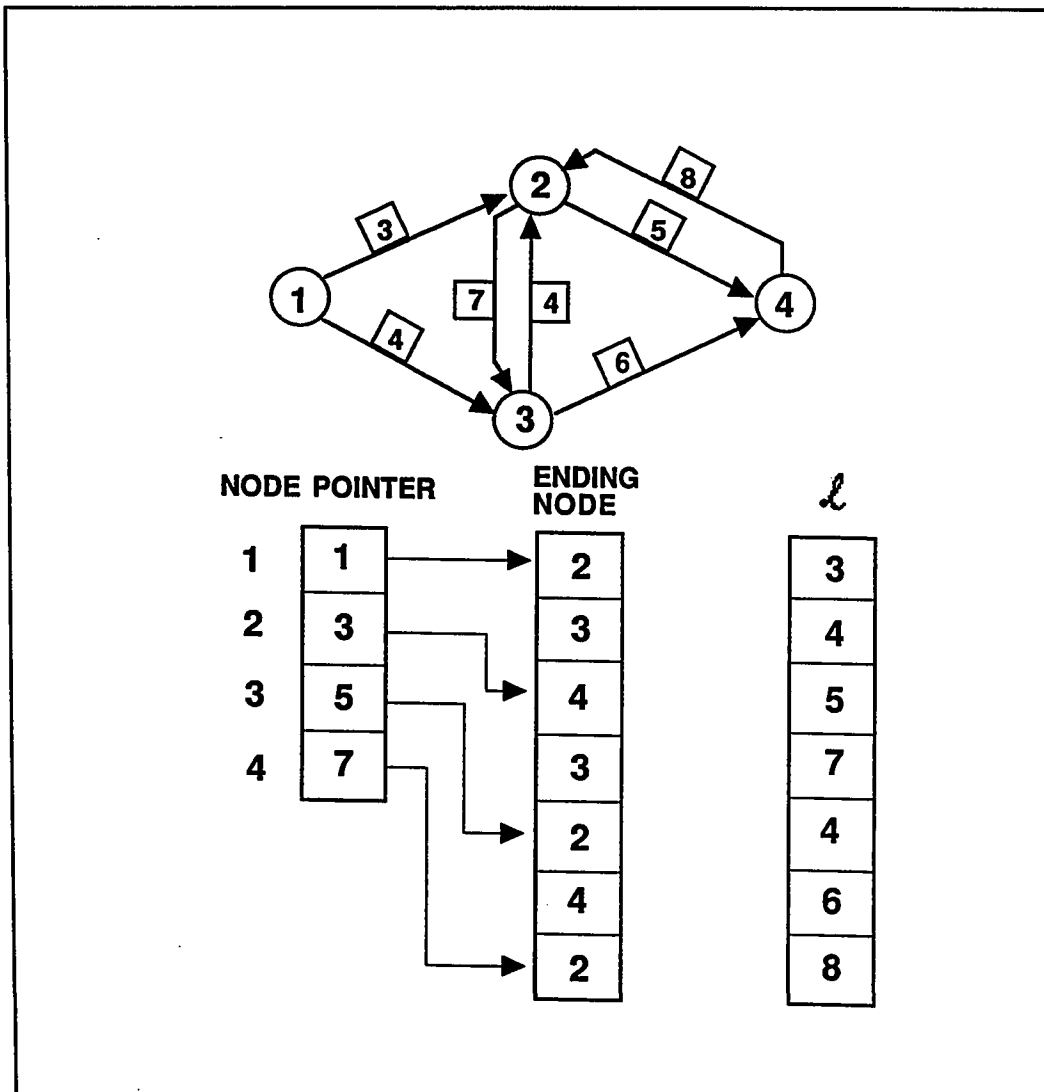


Fig. 2.1. Sorted forward star form (obtained from [24]).

2.3 Literature Review

Linear network flow models are special cases of the linear program. The specialization of the primal simplex algorithm [20, 41] by Dantzig and Johnson [20, 51] is of great importance, since it completely eliminates the need for carrying and updating the basic inverse. In fact, when this specialization is used, the primal simplex method can be performed directly on a network diagram (called, the simplex on a graph algorithm [18]).

Given the importance of linear network flow models many studies have been carried out to develop more efficient algorithms for it [20, 36, 51, 53]. Shortest path, assignment, and transportation problems are important network structured linear programming problems that arise in several papers and contexts and that have deservedly received a great deal of attention in the literature. On the other hand, because of its limited applications, the semi-assignment problem has not received the same attention in the literature. Next, in the following subsections, a review of the literature for the shortest path, assignment, semi-assignment and transportation problems is presented.

2.3.1 Shortest Path Problem

In a network, there are many paths that go from node i to node j , and these paths usually have different lengths. Therefore, the problem of finding the shortest

path from a given node (e.g. r) to all other nodes in a network G may be stated as follows:

Given a network $G(N, A)$ find a minimum tree $T(N_T, A_T)$ of G rooted at node r .

Shortest path analysis is a major analytical component of numerous quantitative transportation and communication models [21, 22, 24, 61]. There are a host of applications for the shortest path algorithm. Edan *et al.* [29] used the shortest path algorithm to find the near optimum locations for fruits picking by a robot. Another application of the shortest path is the median shortest path problem which is the minimization of the path length and the total travel time required for demand to reach a node on the path [19]. Algorithms incorporating shortest path subroutines such as (Enguist 1982 [30], Glover *et al.* 1986 [43], and Hung and Rom 1980 [48]) are coming to challenge network specializations of the primal simplex method.

A diverse set of shortest path models and algorithms have been developed to accommodate these various applications. The major versions of shortest path models are: (i) finding the shortest path from one node to another node, (ii) finding the shortest path from one node (called the root node) to all other nodes, (iii) finding the shortest path from every node to every other node, (iv) finding various types of constrained shortest paths between nodes (e.g., finding the k -th shortest path). Dreyfus [28] has written an excellent paper classifying the types

of algorithms and giving theoretical computational bounds for each class.

Algorithms for the shortest path problem have been developed by many individuals including Bellman (1958), Dijkstra (1959), and Whiting and Hiller (1960), [3, 13, 25]. In 1974, Pape [61] developed a shortest path algorithm and then in [62] he modified his algorithm to be more efficient.

Dijkstra algorithm [25] for finding shortest path to all nodes from a single source node in a network is correct only when all arc weights are nonnegative. However, in 1973 Johnson [50] has modified the Dijkstra algorithms so that it will take care of negative weights.

In 1979, Shir [65] presented several new algorithms for computing K shortest paths in a network. These algorithms utilize strategies which have proved to be efficient in solving shortest path problems.

Recently, Glover *et al.* [39, 40] developed an efficient shortest path algorithm. The algorithm is a polynomially bounded shortest path algorithm, called the partitioning shortest path.

Ahuja *et al.* [1, 2] survey some of the most recent contributions to the field of network flow; and they concentrate on the design and analysis of good algorithms for three core models: the shortest path problem, the maximum flow problem and the minimum cost flow problem. They illustrate some techniques in developing

faster network flow algorithms and their usefulness. In [2] they describe different applications and present a different radix heap implementation of Dijkstra's shortest path algorithm. Gabow and Tarjan [33] have also presented a survey paper on network flow algorithms giving an in-depth account of some of the algorithms discussed in [1, 2].

Dial *et al.* [24] and Klingman *et al.* [56] examined different algorithms for calculating the shortest path from one node to all other nodes in a network. The power of computer implementation and efficient data structures are demonstrated by the efficient method of implementation by Dial and Voorhees [23] for solving the shortest path problem.

Gilsinn and Witzgall [34] as reported by [24] found that improved implementation technology caused solution times for shortest path problems to drop from one minute to slightly more than one second, using the same general shortest path algorithm, computer, and compiler.

Implementation and computational testing in the microcomputer environment of several versions of a threshold shortest path algorithm for finding the shortest path from one node to all other nodes in a network has been done by Klingman and Schneider [57].

Hung and Divoky [47] studied the computational efficiency of five shortest path algorithms. The algorithms include two using threshold functions, two using

heaps, and one using buckets for sorting node labels. They generate three different cost functions to measure the sensitivity of each algorithm to cost distribution. Their results show that the cost distribution does affect algorithm performance differently for different algorithms. Their study showed that threshold algorithms are superior in many cases.

Divoky and Hung [27] examine the way shortest path algorithms perform in a simulated minimum cost network flow problem environment. The algorithms include two using threshold functions, two using heap, and one using buckets for sorting node labels. Empirical computational comparisons with other algorithms are conducted. The results showed that heap and threshold algorithms require more computation time when the number of subtrees increases.

2.3.2 Assignment Problem

An important specialization of the network problem is the assignment problem. The assignment problem is characterized by two sets of n entities, referred to here as jobs and tasks, indexed by i and j , respectively. Each job must be assigned to exactly one task. There is a cost, c_{ij} , for assigning job i to task j , and the objective is to minimize total cost for the n assignments. Mathematically, it can be stated as:

$$\text{Minimize } \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^n x_{ij} = 1, i = \{1, 2, 3, \dots, n\} \quad (2.1)$$

$$\sum_{i=1}^n x_{ij} = 1, j = \{1, 2, 3, \dots, n\}$$

$$x_{ij} \geq 0, \text{ for all } i \text{ and } j$$

Assignment problems arise in numerous applications of flight scheduling, assigning vehicles to routes, project planning, assigning personnel to jobs and a variety of other practical problems in logistics planning. Many studies of assignment problem algorithm have been launched over the past three decades.

Perhaps the best known, most widely, and most written about method for solving the assignment problem is the "Hungarian Method", originally suggested by Kuhn [58]. Carpaneto and Toth [16] have obtained an improvement and varia-

tions in the Hungarian method. The efficiency of their algorithm is mainly due to the pointer technique utilized to locate the unexplored rows and the zero elements of the current matrix. Their algorithm appears to perform better for denser problems.

In the mid-1970's, Barr *et al.* [10] developed a new specialized simplex algorithm, called the alternating path (AP) basis algorithm, which examines only certain basis. It is especially designed to overcome the large number of degenerate pivots in the assignment problem. The AP algorithm was at least twice as fast as previous algorithms. This algorithm turns out to be a special case of the strongly feasible tree algorithm of Cunningham [19] that was independently developed for general network flow problem.

Hung and Rom [48] used the major theoretical results of the alternating path basis algorithm to develop a new algorithm which uses Dijkstra's shortest path algorithm to construct an optimal alternating path basis for a relaxation of the underlying assignment problem. Their algorithm is substantially faster than the alternating path basis algorithm for dense assignment problems.

Balinski and Gomory [6] have described a simple calculation for the assignment which is "dual to" the Hungarian method. While the Hungarian is a dual method, their method is primal and so gives a feasible assignment at each stage of the calculation.

An SSP algorithm for the assignment problem was developed in 1982 by Enguist [30], which is a refinement of the Dinic–Kronrod algorithm [26]. The shortest path algorithm is used to derive a computational very efficient algorithm for solving large, sparse assignment problems. The SSP algorithm goes through a series of modified assignment problems in which some destinations are permitted to have demand greater than one, while some demands are set to zero. The algorithm proceeds from the optimal solution of one of these modified problems to the optimal solution of the next via a related shortest path algorithm. The algorithm terminates when the modified problem coincides with the original assignment problem. The SSP algorithm is found to be three times faster than the AP algorithm and six times faster than the relaxation method by Hung and Rom [48].

Solving the assignment problems as a SSP algorithm also exists in USSR. In 1978, Gribov [45] developed an efficient SSP algorithm which has the property that in the process of solving the original problem a sequence of problems is also solved, each of which is related to a matrix that differs from the preceding by the adjunction of a single row.

The SSP algorithm for the assignment problem was discussed by Glover *et al.* [43]. The threshold shortest path subroutine is used to derive a computational very efficient algorithm. It has been found that this algorithm is seven times faster than the alternating basis algorithm. This algorithm is also discussed in [14].

Balinski [4] suggested another approach based on the dual simplex and known as Signature Method. In each step the signature method goes from one dual feasible basis to a neighboring one. In [5], Balinski developed a competitive dual simplex method for the assignment problem based on the Signature method. Efficient algorithms based upon Balinski's Signature method are described by Goldfarb [44].

In [52] Jonker and Volgenant developed a shortest augmenting path for sparse and dense assignment problems. It is one of the most efficient algorithms for the assignment problem.

Bertsekas [15] developed a new approach for the assignment problem, called the auction algorithm, which assigns jobs to persons using auction. Ahuja et al. [2] suggest a new scaling algorithm for the assignment problem. Their algorithm is based on applying scaling to a hybrid version of the auction algorithm of Bertsekas and the successive shortest path algorithm. The algorithm proceeds by relaxing the optimality conditions, and the amount of relaxation is successively reduced to zero.

The best polynomial algorithm to solve the assignment problem is due to Gabow and Tarjan [33]. This algorithm uses a cost-scaling technique. The scaling algorithm is used for simpler data structures.

Glover *et al.* [36] have done a study of computational time and memory require-

ments of the primal approach for solving the transportation and the assignment problems over the Out-of-Kilter method. They presented an extensive computational experience with special primal simplex algorithm. Then, they compared this algorithm with several Out-of-Kilter computer codes. It has been found that the primal simplex algorithm is roughly twice as fast as the fastest known Out-of-Kilter code (SUPERK). McGinnis [59] presented a computational testing of a primal-dual algorithm. It has been found that the primal simplex methods are more robust because they are insensitive to fluctuations in cost range. The primal-dual methods are sensitive to cost range, but are competitive even when the cost range is as large as 1-10,000. No comparisons exist in the literature between the SSP algorithm, the Hungarian method, the Out-of-Kilter algorithm and the specialized primal simplex algorithm.

2.3.3 Semi-Assignment Problem

The semi-assignment problem is a network problem whose demand constraints are the same as those of an assignment problem and whose supply constraints are the same as those of a transportation problem. Semi-assignment problems arise in numerous applications of scheduling and project planning. Most actual manpower planning and "assignment" problems are semi-assignment problems, and a variety of other practical problems contain embedded semi-assignment problems. Mathematically the semi-assignment problem can be stated as

$$\text{Minimize } \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^n x_{ij} = b_i, \quad i = \{1, 2, 3, \dots, m\} \quad (2.2)$$

$$\sum_{i=1}^m x_{ij} = 1, \quad j = \{1, 2, 3, \dots, n\}$$

$$x_{ij} \geq 0, \quad \text{for all } i \text{ and } j$$

Falling mid-way between the classical assignment and classical transportation problems in its generality, it was bypassed by many researchers in networks. Therefore, this important member of the network family has received scanty attention.

The Out-of-Kilter algorithm, was developed by Fulkerson [32]. The Out-of-Kilter algorithm is not a specialization of a more general method. This algorithm was developed specifically for network programming, and so it can be used to solve the semi-assignment problem.

The algorithms available to solve the transportation problem can solve the semi-assignment problem like the Out-of-Kilter algorithm. Therefore, the specialized primal simplex algorithm developed by Glover *et al.* [30] is one of the best algorithms that can solve the semi-assignment problem.

The first algorithm specially designed for solving the semi-assignment problem was developed by Barr *et al.* [8] in 1977 and was called the alternating path basis method. This algorithm is a specialization of the network simplex method which

exploits the 0–1 flow structures of the semi-assignment problem. They conducted a comparison of the alternative algorithmic approaches using codes for solving the semi-assignment problem. Based on total solution times the alternating path basis was found to be 2.55 times faster than the primal simplex algorithm.

The shortest augmenting path (SAP) algorithm for solving the semi-assignment problem was developed by Kennington and Wang [54] in 1992. The algorithm maintains dual feasibility and complementary slackness and works toward satisfying dual feasibility. An extensive computational comparison of SAP with the best alternative approaches was conducted, and based on the results presented in [54] SAP is uniformly faster than the best competing software for both dense and sparse semi-assignment problems having a size up to 80,000 arcs.

2.3.4 Transportation Problem

In the direct sense, the transportation model seeks the determination of a transportation plan of a single commodity from a number of sources to a number of destinations. The data of the model include:

1. Level of supply at each source and amount of demand at each destination.
2. The unit transportation cost of the commodity from each source to each destination.

Mathematically, the transportation problem can be formulated as

$$\text{Minimize } \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^n x_{ij} = a_i, \quad i = \{1, 2, 3, \dots, m\} \quad (2.3)$$

$$\sum_{i=1}^m x_{ij} = b_j, \quad j = \{1, 2, 3, \dots, n\}$$

$$x_{ij} \geq 0, \quad \text{for all } i \text{ and } j$$

for a balanced transportation problem $\sum_i a_i = \sum_j b_j$.

Transportation problems are perhaps the most visible in our everyday lives. The traditional application of the transportation problem is illustrative. In the transportation problem, a shipper with inventory of goods at its warehouses must ship these goods to geographically dispersed retail centers, each with a given customer demand, and the shipper would like to meet these demands incurring the minimum possible transportation costs.

Hitchcock [46] was credited with the first formulation and discussion of a transportation model. Then, Dantzig [20,22] adapted his simplex method to solve transportation problems. Dantzig method is known as Row-Column Sum Method.

Charnes and Cooper [17] developed an intuitive presentation of Dantzig's procedure through what is called the "stepping stone" method.

Balinski and Gomory [6] developed a simple calculation for the transportation problem which is “dual to” the well known Hungarian method. This method is primal and gives bounds on the number of steps required for the transportation problem.

Barr *et al.* [11] developed a new primal extreme point algorithm for solving capacitated transportation problems. The algorithm is called the generalized alternating path (GAP) algorithm. The GAP algorithm is specifically designed to take advantage of the often pervasive primal degeneracy of transportation problems.

Barr *et al.* [12] presented a new specialized simplex algorithm for solving capacitated transshipment network problems. The method exploits the combinatorial possibilities available in degeneracy to obtain significant theoretical and computational advances.

Solving the transportation problem as a SSP algorithm has been developed by Gribov [45]. The algorithm has the property that in the process of solving the original problem a sequence of problems are also solved, each problem is related to a matrix that differs from the preceding problem by the adjunction of a single row.

Glover *et al.* [36] developed primal and dual simplex algorithms for general network flow problems. Testing these algorithms against the best Out-of-Kilter code (SUPERK) showed that the primal code is roughly two times faster.

Glover *et al.* [37] presented an in-depth computational comparison of the basic solution algorithm for solving transportation problems. The comparison is performed using computer codes for the dual simplex transportation method, the Out-of-Kilter method, and the primal simplex transportation method (Row-Column Sum Method).

Glicksman *et al.* [35] coded the primal simplex method for the transportation problem. The technique she used has been found useful in simplifying coding and in reducing solution time on a computer.

2.4 Conclusion

This chapter reviewed network terminology and storage schemes used to represent a network in a computer memory. From the review of the literature, it can be concluded that no algorithms for the semi-assignment and transportation problems have been developed using the concept developed by Enguist [25]. Since the SSP algorithm developed by Enguist is one of the fastest algorithms for solving the assignment problem, and there is a need for an efficient and fast algorithm for the semi-assignment and transportation problems, it is felt that there is a need to extend the SSP algorithm to solve these problems. Then test and compare the developed algorithms with the most efficient ones in the literature for these problems. This is the subject of the next chapter.

CHAPTER 3

AN EFFICIENT SUCCESSIVE SHORTEST PATH ALGORITHM FOR THE ASSIGNMENT PROBLEM

3.1 Introduction

Successive shortest path (SSP) approach has been proposed for solving the assignment problem [30]. The heart of the SSP algorithm is the shortest path routine used for solving the resulting shortest path problems. Therefore, the efficiency of the SSP algorithm depends to a great extent on the choice of the shortest path method. In [30] Dijkstra algorithm has been implemented in a SSP algorithm for solving the assignment problem.

The purpose of this chapter is to identify the best shortest path method that results in an efficient implementation of the SSP algorithm for the assignment problem presented in [30]. The shortest path algorithms tested are Dijkstra [25], Dantzig [21] and Pape [61]. Then the most efficient SSP algorithm is compared with the best methods known for solving the assignment problem. The best methods used in the comparisons are: the Hungarian method, a specialized version of the Simplex method and Out-of-Kilter algorithm. The code for the Hungarian method is obtained from [16], and for the Simplex method obtained from the

University of Texas at Austin, a commercial code known as CAPNET [6]. The Out-of-Kilter code is a package in Statistical Analysis System (SAS) software [63].

The rest of the chapter is organized as follows: Section 3.2 describes the steps of the SSP algorithm. Section 3.3 reviews the three shortest path algorithms used in the testing and outlines efficient data structures for implementing them. Computational aspects and the design of the tests are discussed in section 3.4. Section 3.5 presents the comparisons between the shortest path algorithms in order to identify the best one that fits the SSP approach. Section 3.6 compares the most efficient SSP algorithm for the assignment problem with the best known methods for solving the assignment problem. The comparisons are based on average of central processing unit (CPU) time and memory requirements.

3.2 Description of the SSP Algorithm for the Assignment Problem

A mathematical model for the assignment problem is given by the following:

$$\begin{aligned}
 &\text{Minimize} && \sum_{(i,j) \in E} c_{ij} x_{ij} \\
 &\text{subject to} && \\
 &&& \sum_{(i,j) \in FS(i)} x_{ij} = 1, i \in I = \{1, 2, 3, \dots, n\} && (3.1) \\
 &&& \sum_{(i,j) \in RS(j)} x_{ij} = 1, j \in J = \{1, 2, 3, \dots, n\} \\
 &&& x_{ij} \geq 0, (i, j) \in E
 \end{aligned}$$

where I is the set of origin nodes, J is the set of destination nodes, E is the set of arcs, c_{ij} is the cost of a unit flow on arc (i, j) , $FS(i)$ is the forward star of node i and $RS(j)$ is the backward star for node j .

Next, we introduce a concept that is central to the development of the SSP algorithm. Whenever a mapping $A : I \rightarrow J$ is given, then A defines a *tentative assignment* provided that $(i, A_i) \in E$ for $i \in I$, and A_i is the image of i under this mapping.

Since the assignment network will remain fixed in the following discussion, denote the assignment problem equipped with a tentative assignment A by (C, A) . (C, A) is in *standard form* if $c_{ij} \geq 0$ for all $(i, j) \in E$ and $c_{ij} = 0$ when $j = A_i$. When (C, A) is in standard form and A is one-to-one, then A determines an

optimal solution of the assignment problem [30].

In the starting procedure for SSP a tentative assignment A is defined as follows.

First,

$$\hat{c}_i = \min_{(i,p) \in FS(i)} \{c_{ip}\} \quad (3.2)$$

must be determined for $i \in I$. Next, for $i \in A$, A_i is defined to be some j such that $c_{ij} = \hat{c}_i$. For this A , (C, A) may not be in standard form [30]. However, the forward star of each origin i may be scaled by setting

$$c_{ip} \leftarrow c_{ip} - \hat{c}_i \quad (3.3)$$

for $(i,p) \in FS(i)$. The resulting (C, A) is in standard form [30] This technique is also used in the starting procedure for the SSP algorithm. Such scaling does not affect the solution of the original assignment problem.

For a given tentative assignment A , let a_j denote the number of elements in $\{i : j = A_i\}$.

The modified assignment problem relative to (C, A) is defined as follows:

$$\begin{aligned}
& \text{Minimize} && \sum_{(i,j) \in E} c_{ij} x_{ij} \\
& \text{subject to} && \\
& && \sum_{(i,j) \in FS(i)} x_{ij} = 1, \quad i \in I && (3.4) \\
& && \sum_{(i,j) \in RS(j)} x_{ij} = a_j, \quad j \in J \\
& && x_{ij} \geq 0, \quad (i,j) \in E
\end{aligned}$$

When (C, A) is in standard form, A provides an optimal solution to the modified assignment problem relative to (C, A) .

A destination node is said to be *abundant* relative to A when $a_j > 1$. Likewise, j is said to be *deficient* relative to A when $a_j = 0$.

Suppose that (C, A) is in standard form and d is some deficient node with respect to A . Then the shortest path problem relative to (C, A) and d is denoted $SP(C, A, d)$ and is defined as follows. The network for $SP(C, A, d)$, which is referred to as the shortest path network, is derived from the assignment network. We proceed by describing how this is done and how the arc lengths for $SP(C, A, d)$ are defined. The nodes of the shortest path network can be identified with arcs (i, j) of the assignment network that satisfy $j = A_i$. Such a shortest path node is denoted by $(i \rightarrow A_i)$ or $(i \rightarrow j)$, where $j = A_i$. Also, it is referred to $(i \rightarrow A_i)$ as the i -th shortest path node. Clearly, there are n such nodes. Introduce one more node $(n + 1 \rightarrow d)$ for the shortest path network and make this consistent

with previous notation by extending A so that $A_{n+1} = d$. For $SP(C, A, d)$, the root node is $(n + 1 \rightarrow d)$, while a node $(i \rightarrow A_i)$ is abundant, provided A_i is an abundant destination relative to A . An arc exists in the shortest path network from $(i \rightarrow A_i)$ to $(p \rightarrow A_p)$ in case $(p, A_i) \in E$. If this arc exists, its length is c_{pj} where $j = A_i$.

The major steps of the SSP algorithm will be given below after the following notations are defined.

R_i = node potential for the i -th origin node

K_j = node potential for the j -th destination node

D_i = distance of the i -th shortest path node from the root

P_i = predecessor of the i -th shortest path node in the shortest path tree.

The use of R is to denote the mapping whose value at i is R_i . The use of K, D and P is similar. When a shortest path problem is solved, denote the first abundant node to be permanently labelled by v , and denote the distance of v from the root by L . Let $S(v)$ denote a unique sequence of nodes beginning with a node $v \neq r$ and leading to r in a shortest path tree. Let $C^0 = \{c_{ij}^0 : (i, j) \in E\}$ denote the costs of the origin (unmodified) assignment problem. Then the steps of the SSP algorithm are:

0. Define A^1 and transform C^0 to C^1 by scaling as described in equation (3.3) above so that (C^1, A^1) is in standard form. Set $k = 1$.

1. Choose a destination node d^k which is deficient relative to A^k . If no deficient nodes exist, stop, A^k defines an optimal solution.
2. Solve $SP(C^k, A^k, d^k)$. The shortest path algorithm is terminated as soon as an abundant shortest path node is permanently labelled. If the shortest path algorithm fails to permanently label an abundant node, stop, the assignment problem is infeasible. Otherwise, the results of this step are D^k, P^k, v^k and L^k .
3. For each permanently labelled shortest path node $(i \rightarrow j)$ from step 2, set $R_i^k = D_i^k - L^k$ and $K_j^k = L^k - D_j^k$. For any remaining origins i or destinations j , set $R_i^k = K_j^k = 0$.
4. Set $c_{ij}^{k+1} = c_{ij}^k - R_i^k - K_j^k$ for $(i, j) \in E$.
5. Whenever the i -th shortest path node is in $S(v^k)$, $i \neq n+1$, set $A_i^{k+1} = A_\ell^k$ where $\ell = P_i^k$. For all other origins, i , $A_i^{k+1} = A_i^k$. Set $k \rightarrow k+1$ and go to step 1.

The theoretical properties of the above algorithm are shown in [30]. It has been shown that the algorithm converges to the optimal solution in $O(n^3)$ computational bound. Also, an example is given in [30] to demonstrate and fix the idea of the SSP algorithm for the assignment problem.

The shortest path problem $SP(C, A, d)$ in step 2 can be solved using many algorithms developed for solving the shortest path problem. In the following sec-

tion we introduce three shortest path algorithms (Dijkstra, Dantzig and Pape) and outline efficient data structures for implementing them. They will be used in the SSP framework in order to identify the most efficient one that fits the SSP algorithm.

To clarify the steps of the algorithm, the following example is presented. The example shows how $SP(C, A, d)$ is defined in a particular case. Let the assignment problem be as shown in Figure 3.1, where arc (i, j) is labelled with cost c_{ij} .

Step 0

Put the problem into standard form:

$$\hat{c}_1 = 2, \hat{c}_2 = 3, \hat{c}_3 = 1$$

$$c_{11}^1 = c_{11}^0 - \hat{c}_1 = 2 - 2 = 0; \quad c_{32}^1 = c_{32}^0 - \hat{c}_3 = 1 - 1 = 0$$

$$c_{12}^1 = c_{12}^0 - \hat{c}_1 = 3 - 2 = 1; \quad c_{33}^1 = c_{33}^0 - \hat{c}_3 = 2 - 1 = 0$$

$$c_{21}^1 = c_{21}^0 - \hat{c}_2 = 3 - 3 = 0$$

$$c_{22}^1 = c_{22}^0 - \hat{c}_2 = 4 - 3 = 1$$

$$c_{23}^1 = c_{23}^0 - \hat{c}_2 = 5 - 3 = 1$$

Therefore,

$$A^1 = (1, 1, 2) \text{ and}$$

$$a_1^1 = 2, a_2^1 = 1, a_3^1 = 0$$

$$k = 1.$$

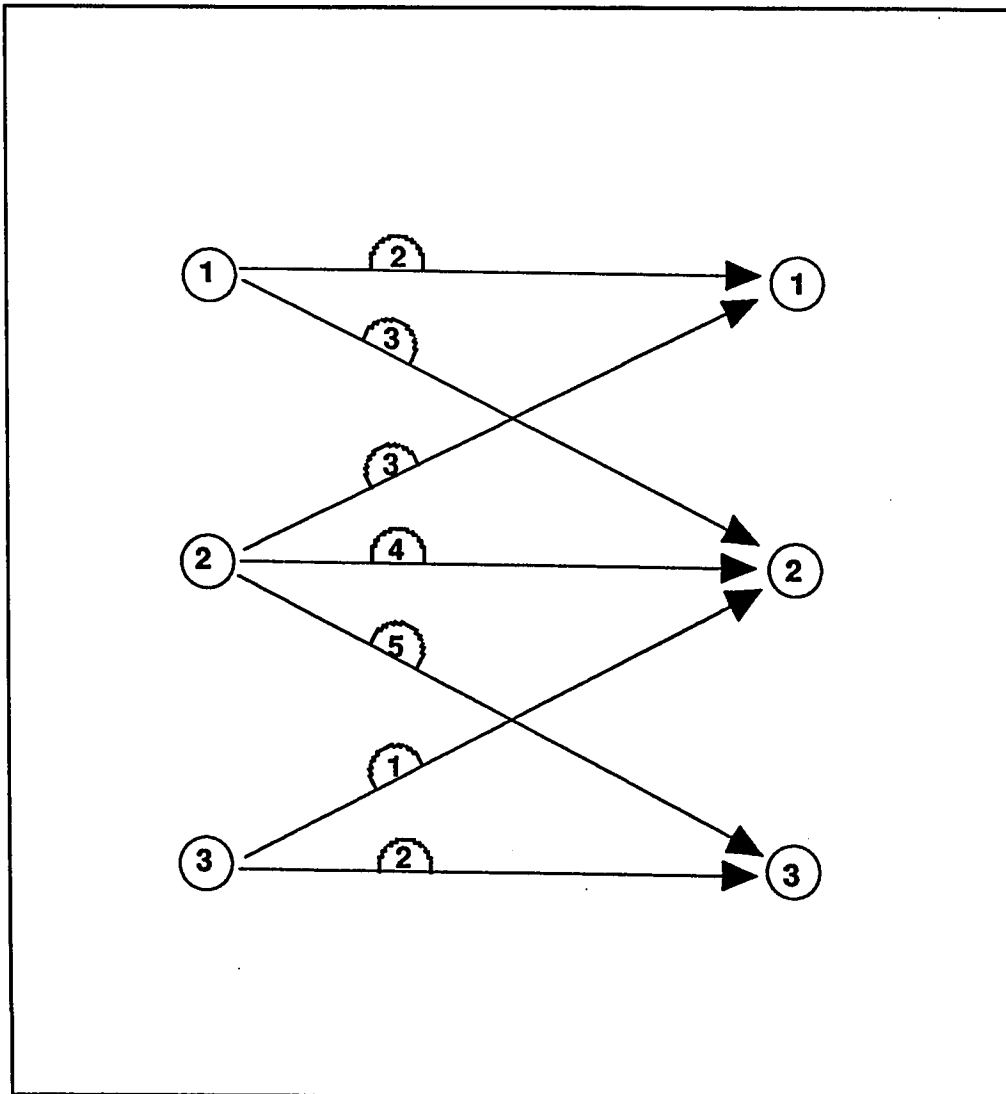


Figure 3.1 An Example of an Assignment Problem

Iteration 1.

Step 1: Choose a deficient ode, $d^1 = 3$.

Step 2: Solve $SP(C^1, A^1, d^1)$.

The network for $SP(C^1, A^1, d^1)$ is shown in Figure 3.2 where a shortest path node ($i \rightarrow A_i$) is shown as a node with an upper label (i) and a lower label ($A_i = j$).

The arcs of the shortest path network are labelled with their lengths.

The result of this step are:

$$D_1^1 = 1, P_1^1 = 3 \quad v^1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ and}$$

$$D_3^1 = 0, P_3^1 = 4 \quad L^1 = 1$$

Step 3:

$$R_1^1 = D_1^1 - L^1 = 1 - 1 = 0; K_1^1 = L^1 - D_1^1 = 1 - 1 = 0$$

$$R_3^1 = D_3^1 - L^1 = 0 - 1 = -1; K_2^1 = L^1 - D_3^1 = 1 - 0 = 1$$

$$R_2^1 = K_3^1 = 0$$

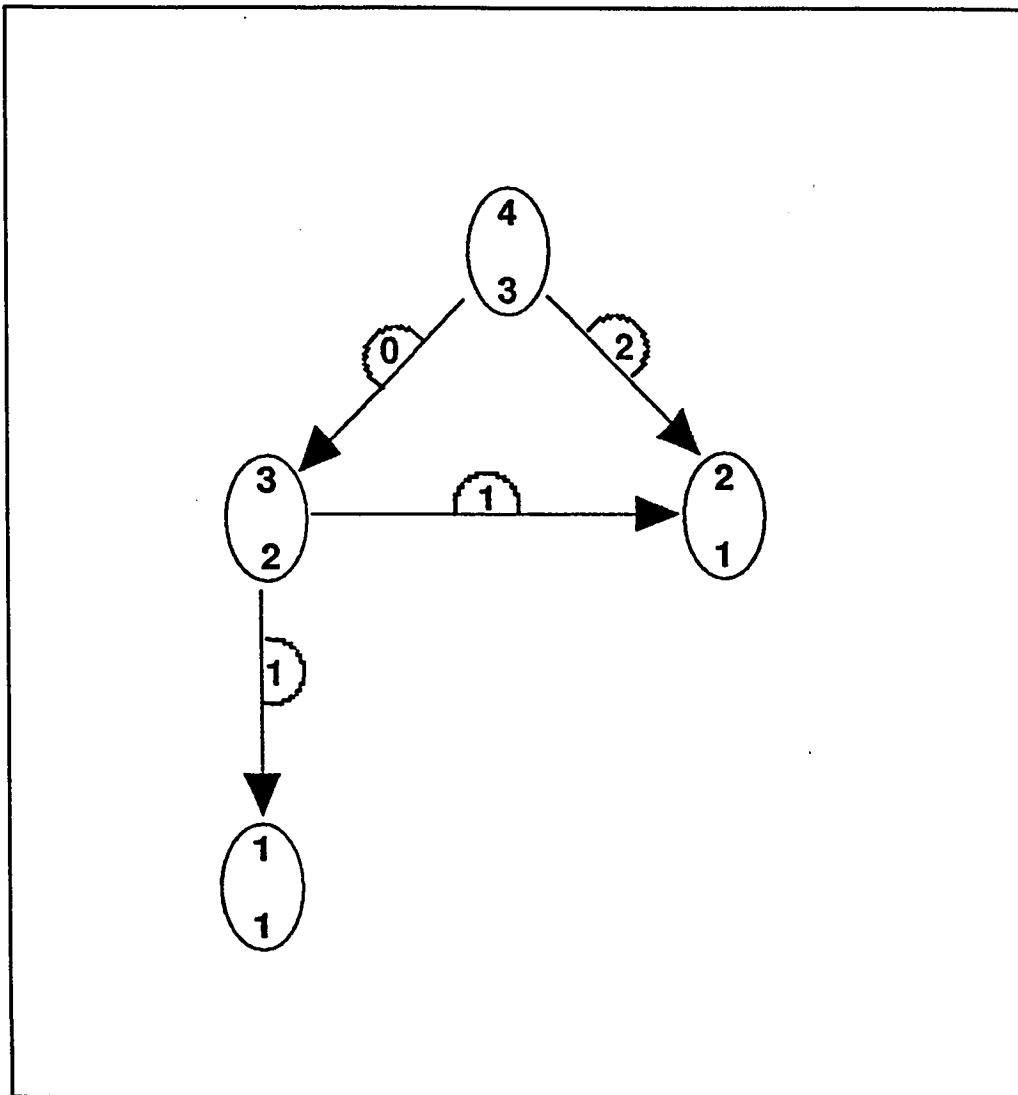


Figure 3.2 An Example of $SP(C, A, d)$

Step 4:

$$C_{11}^2 = C_{11}^1 - R_1^1 - K_1^1 = 0 - 0 - 0 = 0; C_{12}^2 = C_{12}^1 - R_1^1 - K_2^1 = 1 - 0 - 1 = 0$$

$$C_{21}^2 = C_{21}^1 - R_2^1 - K_1^1 = 0 - 0 - 0 = 0; C_{22}^2 = C_{22}^1 - R_2^1 - K_2^1 = 1 - 0 - 1 = 0$$

$$C_{23}^2 = C_{23}^1 - R_2^1 - K_3^1 = 2 - 0 - 0 = 2$$

Step 5:

$$A^2 = (2, 1, 3)$$

$$a_1^2 = 1, a_2^2 = 1, a_3^2 = 1$$

Let $K = 2$, go to step 1

Iteration 2

Step 1: No deficient node, stop.

3.3 The Shortest Path Problem and Labelling Methods

The problem of finding the shortest paths from a given node r to all other nodes in network $G(N, A)$ may be stated as that of finding a minimum tree $T(N_T, A_T)$ of G rooted at node r . The best methods for solving the shortest path problem are labelling methods, which are reviewed in the next subsections.

3.3.1 Labelling Methods

Labelling methods for computing such a minimum tree have been divided into two general classes, label-setting and label-correcting methods. Both methods typically start with a tree $T(N_T, A_T)$ such that $N_T = \{r\}$ and $A_T = \phi$. A label-setting method then augments N_T and A_T respectively, by one node $v \in N$ and one arc $(u, v) \in A$ at each iteration in such a manner that $u \in N_T$, $v \notin N_T$, and the unique path from r to v in T is a shortest path. A label-setting method terminates when all arcs in A which have their starting endpoints in N_T also have their ending endpoints in N_T .

A label-correcting method, on the other hand, always exchanges, augments, or updates arcs in A_T in a manner that replaces or shortens the unique path from r to v in T , but does not guarantee that the new path is a shortest path (until termination occurs). Using the notation defined in section 2.2, we now give a precise description of each of these *general* methods.

3.3.1.1 General Label-Setting Method

1. Initialize a tree $T(N_T, A_T)$ such that $N_T = \{r\}$ and $A_T = \phi$. Further, set $p(t) := 0$, $t \in N$; $d(t) := \infty$, $t \in N - \{r\}$; and $d(r) := 0$. (The notation $a := b$ sets a equal to b .)
2. Let $S = \{(u, v) : u \in N_T; v \in N - N_T, (u, v) \in A\}$. If $S = \phi$, go to step 4. Otherwise proceed.
3. Let $d(u) + \ell(u, v) = \min_{(p, q) \in S} (d(p) + \ell(p, q))$. Redefine

$$N_T := N_T \cup \{v\}$$

$$A_T := A_T \cup \{(u, v)\}$$

$$p(v) := u$$

$$d(v) := d(u) + \ell(u, v)$$

and repeat step 2.

4. Stop. $T(N_T, A_T)$ is a minimum tree and for each node $v \in N$, $d(v)$ is the length of a shortest path from r to $v \neq r$.

It is worth noting that a label-setting method only works for non-negative arc length. A label-correcting method, however, works for negative arc lengths as long as there are no circuits of negative length in the network $G(N, A)$.

3.3.1.2 General Label-Correcting Method

1. Initialize a tree $T(N_T, A_T)$ such that $N_T = \{r\}$ and $A_T = \phi$. Further, set $p(t) := 0, t \in N; d(r) := 0$; and $d(t) := \infty, ; t \in N - \{r\}$.
2. Go to step 4 if there does not exist an arc $(u, v) \in A$ such that $d(u) + \ell(u, v) < d(v)$. Otherwise, for such an arc, redefine

$$N_T := N_T \cup \{v\}$$

$$A_T := A_T - \{(s, v) \in A_T\} \cup \{(u, v)\}$$

$$p(v) := u$$

$$d(v) := d(u) + \ell(u, v)$$

3. Repeat step 2.
4. Stop. $T(N_T, A_T)$ is a minimum tree and for each node $v \in N, d(v)$ is the length of a shortest path from r to $v \neq r$. Further, if a shortest path from r to v exists (i.e., if $p(v) \neq 0$), then it may be constructed by successively examining the predecessors of v until the root node r is encountered.

3.3.2 Implementation Techniques for the Label-Setting Method

The implementation techniques and data structures for label-setting method prepared by Dijkstra and Dantzig are obtained from [24] and presented in appendix A. This includes Dijkstra and Dantzig address calculation sort lists.

3.3.3 Implementation Technique for the Label-Correcting Method

In this subsection, we discuss an implementation of the general label-correcting algorithm. The algorithm is Pape algorithm which is a modification of Moore's algorithm [61, 63].

3.3.3.1 Pape Method

The main idea of this algorithm centers around the "status" of a new-found successor node j . If j has not yet been reached by the process, it is entered at the end of the successor list; if it is currently in the list, no new entry is made, but if it has already been processed and removed from the list, it is entered at the top of the list so that it will be processed next. The current status of each node is recorded by the values of the list which are stored as two-way linked list.

To implement this algorithm, the algorithm initializes $P(v) = 0$, $v \in N$; $d(r) = 0$ and $d(v) = \infty$, $v \in N - \{r\}$; and $B(1) = r$ and $B(i) = \phi$, $2 \leq i \leq N$, where

B is the sequence list of length = N . The root node r is then scanned and the improving node of $FS(r)$ is "added to" the two-way linked list of B . If this node has not yet been reached then enter it at the end of the list. If it has already been processed, then enter it at the top of the list. The first pass of the B list starts at the top, examining the elements of B . Each node v associated with this element is then either added or not added to the list. Then node v will be removed from the top of the list and it will never be added to B and thus no steps are required to remove it.

The great advantage of the above-mentioned technique is that errors in minimal distance are often corrected as soon as they are detected and not allowed to progress further. In practice, this leads to a considerable reduction in CPU time, particularly for grid networks.

3.4 Computational Aspects and Experimental Design

Perhaps the most difficult aspect of algorithm development is empirical evaluation, primarily because of the lack of standards for comparison. It is widely recognized that it is simply not valid to compare algorithms across different problems sets, different implementations and different computing environments.

In the following subsections we discuss some comments regarding the implementation of SSP algorithm and computing environments.

3.4.1 Computational Aspects

We have developed a FORTRAN Codes called SPAN-I, SPAN-II and SPAN-III, which are an implementation of the SSP algorithm. The only difference between the three codes is in step 2. SPAN-I solves the shortest path problem using Dijkstra algorithm, SPAN-II solves the shortest path problem using Dantzig algorithm and SPAN-III solves the shortest path problem using Pape algorithm. In this subsection, we will discuss some of the details of these implementations.

Let,

$$R_i^0 = \min_{(i,j) \in FS(i)} \{c_{ij}^0\}, i \in I, K_j^0 = 0, j \in J$$

$$R_i^{-m} = \sum_{k=0}^{m-1} R_i^k, i \in I, K_j^{-m} = \sum_{k=0}^{m-1} K_j^k, j \in J$$

R_i^{-m} and K_j^{-m} are the accumulated node potentials for the modified assignment problem relative to (C^0, A^m) .

The major steps of SSP listed in section 3.2 were formulated for ease of exposition and not for computational efficiency. For this reason, there is a difference between steps 3 and 4 as listed and what is done in the codes. In step 3 node potentials are defined for all nodes of the assignment network at each iteration, while in the codes, the accumulated node potentials are maintained. Thus, at iteration k , only the node potentials corresponding to permanently labelled shortest path nodes need to be updated. In step 4 the cost data for all arcs is updated; however, this is not done in the codes. Instead, whenever a cost c_{ij}^k is needed in the solution of $SP(C^k, A^k, d^k)$, it is computed using the relation $c_{ij}^k = c_{ij}^0 - R_i^{-k} - K_j^{-k}$. Next, we describe how the details of some SSP steps were handled in the codes.

In step 1 of SSP, there may be more than one d^k that could be chosen. Enguist [25] did some experimentation but was unable to develop a more efficient strategy than simply choosing the smallest j such that j is deficient. This is the strategy we used in the codes.

As mentioned in section 3.3.2.1, the length of the sort list employed in solving

the shortest path problem $SP(C^k, A^k, d^k)$ using Dijkstra algorithm depends on the maximum arc length in the problem. Making a complete pass through the arc data to determine the maximum shortest path arc length at each iteration of SSP would be very inefficient. In the codes, we simply maintain an upper bound on the maximum shortest path arc length and use this upper bound in determining the length of the sort list or the size of the radix. if we let

$$\bar{c} = \max_{(i,j) \in E} \{c_{ij}^0\}$$

and

$$\hat{K}^k = \min_{j \in J} \{K_j^k\}$$

then the upper bound on arc lengths for $SP(C^k, A^k, d^k)$ is $\bar{c} - \hat{K}^k$. This upper bound is easily updated along with the node potential K^{-k} .

Using Dantzig algorithm instead of Dijkstra algorithm for solving the shortest path problem in step 2 is expected to take more time because in Dantzig each time we scan a node we need to sort the array k . A modification to the algorithm will minimize the execution time. Instead of sorting, we created an array containing the c_{ij} , each time a node wants to be scanned we search for the minimum c_{ij} if we find one we put $c_{ij} = \infty$ and we take arc $(i \rightarrow j)$ as an improving arc. Then we proceed with the SSP algorithm.

There is a final comment we would like to mention regarding adapting the Pape algorithm (label-correcting) for SSP. Pape algorithm will not stop until all the arcs

are scanned, this will take a lot of time and it will not fit with SSP framework, because in SSP once an abundant node is permanently labelled the shortest path algorithm is terminated. Therefore, finding an abundant node is not enough it should be permanently labelled. Because, Pape algorithm does not guarantee that this path is the shortest path to the abundant node, what has been done is the following, when an abundant node is found the reverse star of this node is checked and the predecessors of it until we reach the root node. If all the arcs entering these nodes are scanned this guarantees that this node is permanently labelled.

3.4.2 Experimental Design

All of the developed codes are in-core codes, *i.e.*, the program and all of the problem data simultaneously reside in fast-access memory. They are all coded in FORTRAN IV and compiled using the same compiler transactions and same computer (AMDHAL 5850 with speed 9.3 million transactions per second and with 32 M bytes of shared memory). The computer jobs were executed during periods when the machine load was approximately the same, and all solution times are exclusive of input and output. The total time spent solving the problem was recorded by calling a Real Time Clock upon starting to solve the problem and again when the solution was obtained. Each test problem is solved five times and the average solution time is reported.

The problems used in the tests were randomly generated using FORTRAN code, called NETGEN [55]. The code "NETGEN" can generate capacitated and uncapacitated transportation and minimum cost flow network problems, and assignment problems. In addition to generating structurally different classes of network problems the code permits the user to vary structural characteristics within a class. The assignment problems generated has been used previously in computational testing in [10, 30, 59]. We added some problems like the sizes 50×50 and 150×150 for more elaborate comparisons. The sizes of the problems were 50×50 , 100×100 with arc densities of 25%, 50%, 75% and 100%, 150×150 with arc densities of 4.4%, 13.3%, 22.2%, 31.1% and 40%, and 200×200 assignment problems with arc densities of 3.75%, 5.625%, 7.5%, 9.375%, and 11.25%.

The cost coefficients are randomly generated between 1 and 100 in the first run with the same data generated using NETGEN. Then the same problems are run using another cost coefficient range between 1 and 10000. The reason for this design is because some algorithms can behave differently when the range of cost coefficients varies [48].

3.5 Comparison of Dijkstra, Dantzig and Pape Algorithms for Solving the Shortest Path Problem in the SSP Algorithm

Each one of the developed codes (SPAN-I, SPAN-II and SPAN-III) can solve the assignment problem. To evaluate the efficiency of each code, their solution times are compared for the same set of problems. Another important aspect of evaluating an algorithm is its efficiency in storing the data (*i.e.* the amount of memory required to store the data).

Tables 3.1, 3.2, and 3.3 show the size of the problems, number of arcs in each problem, the cost range and the solution times in seconds for the three codes. The solution time is an average of five runs for each problem. Graphical representation of the solution times are also presented in Figures 3.3 to 3.5.

Based on total solution times for the problems shown in Tables 3.1, 3.2 and 3.3, SPAN-I is about 2.4 times faster than SPAN-II and roughly nine times faster than SPAN-III. Therefore, in terms of CPU time SPAN-I is more efficient than SPAN-II and SPAN-III. This leads to the conclusion that Dijkstra implementation for the label-setting is the appropriate method that fits the SSP approach for the assignment problem.

Next, we compare the number and size of arrays required by the codes. Table 3.4 shows the codes memory requirements.

TABLE 3.1

Solution Times in Seconds for 50×50 Assignment Problems

Number of Arcs	Cost Range	Time (seconds)		
		SPAN-I	SPAN-II	SPAN-III
625	1 - 100	0.0084	0.0392	0.0871
1250	1 - 100	0.0273	0.0694	0.1943
1875	1 - 100	0.0293	0.0982	0.2582
2500	1 - 100	0.0318	0.1366	0.5067
625	1 - 10000	0.0338	0.1176	0.1525
1250	1 - 10000	0.0946	0.1503	0.3419
1875	1 - 10000	0.1250	0.2700	0.5621
2500	1 - 10000	0.1796	0.3688	0.8117
Total solution Times		0.5298	1.2501	2.9145
Ratio		1	2.36	5.50

SPAN-I = using Dijkstra algorithm

SPAN-II = using Dantzig algorithm

SPAN-III = using Pape algorithm

TABLE 3.2

Solution Times in Seconds for 100×100 Assignment Problems

Number of Arcs	Cost Range	Time (seconds)		
		SPAN-I	SPAN-II	SPAN-III
2500	1 - 100	0.0752	0.2470	0.9301
5000	1 - 100	0.1171	0.3029	1.283
7500	1 - 100	0.1972	0.4218	1.960
10000	1 - 100	0.2187	0.4873	2.829
2500	1 - 10000	0.0937	0.3587	1.341
5000	1 - 10000	0.2132	0.3739	1.965
7500	1 - 10000	0.2203	0.4517	2.845
10000	1 - 10000	0.2874	0.5133	3.461
Total Solution Times		1.4228	3.1566	16.6141
Ratio		1	2.21	11.6

SPAN-I = using Dijkstra algorithm

SPAN-II = using Dantzig algorithm

SPAN-III = using Pape algorithm

TABLE 3.3

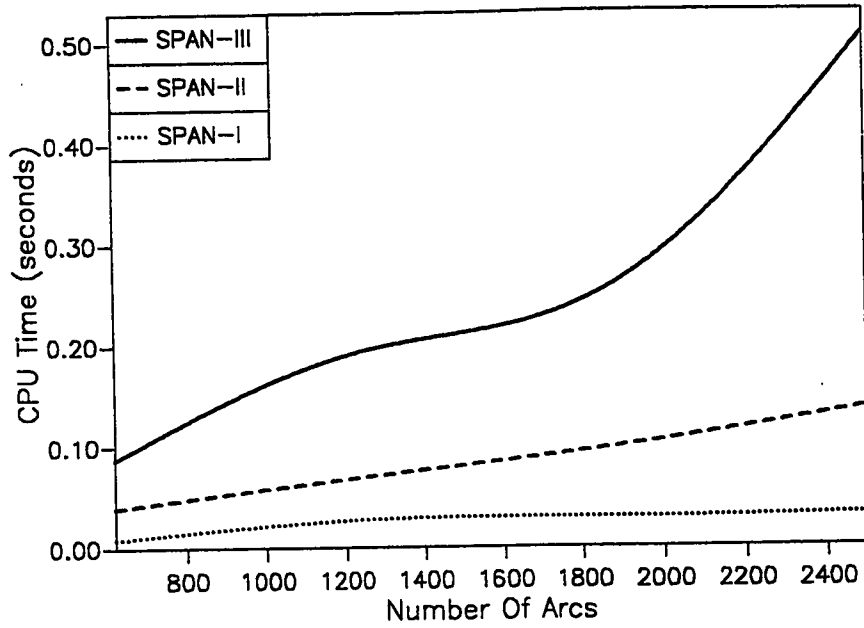
Solution Times in Seconds for 200×200 Assignment Problems

Number of Arcs	Cost Range	Time (seconds)		
		SPAN-I	SPAN-II	SPAN-III
1500	1 - 100	0.0533	0.3109	1.006
2250	1 - 100	0.1201	0.3508	1.294
3000	1 - 100	0.1631	0.4272	1.407
3750	1 - 100	0.2154	0.4963	1.467
4500	1 - 100	0.2054	0.5744	1.603
1500	1 - 10000	0.1058	0.4253	1.232
2250	1 - 10000	0.1604	0.4428	1.451
3000	1 - 10000	0.1830	0.5782	1.683
3750	1 - 10000	0.2495	0.6103	1.708
4500	1 - 10000	0.3673	0.6904	2.172
Total Solution Times		1.8233	4.9066	15.023
Ratio		1	2.69	8.23

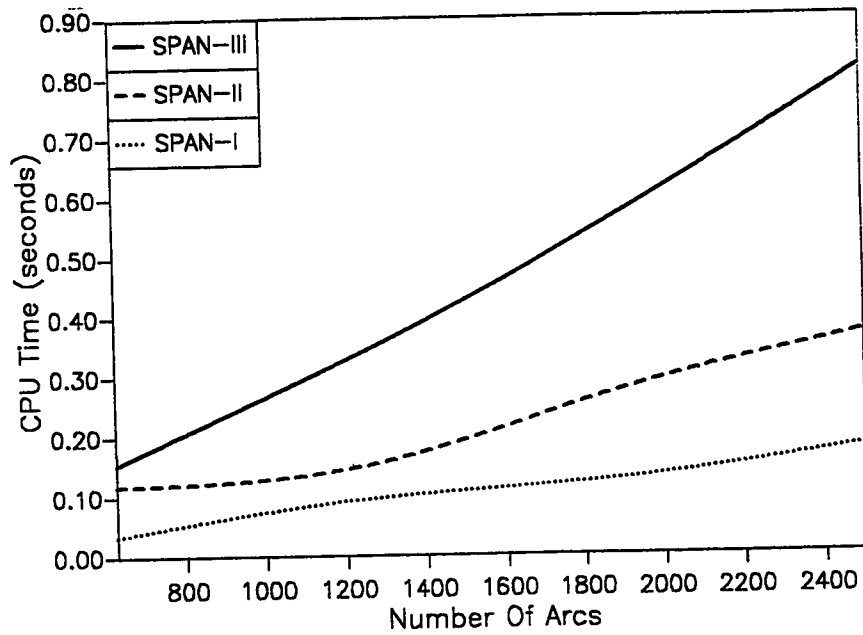
SPAN-I = using Dijkstra algorithm

SPAN-II = using Dantzig algorithm

SPAN-III = using Pape algorithm

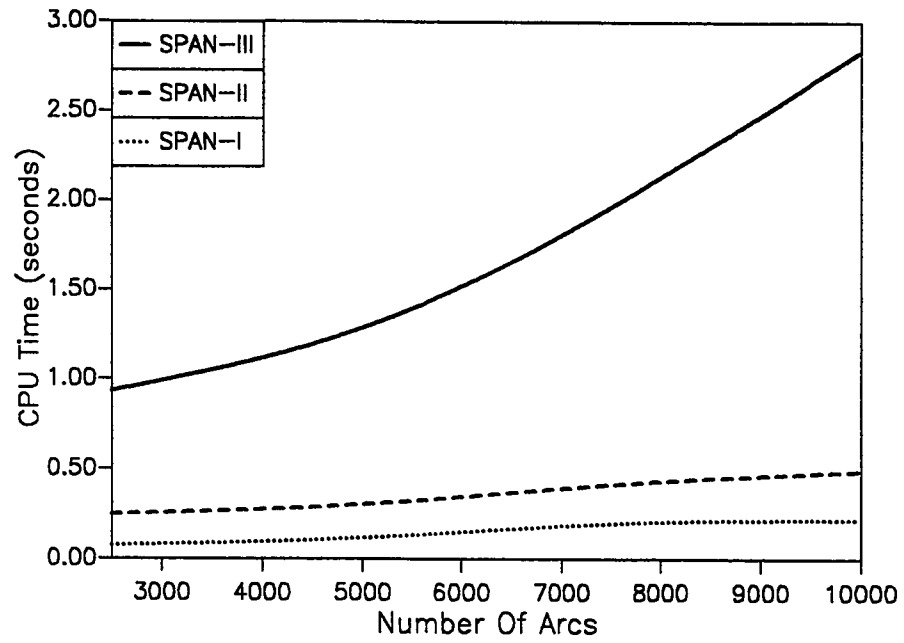


a. Cost Range 1 - 100

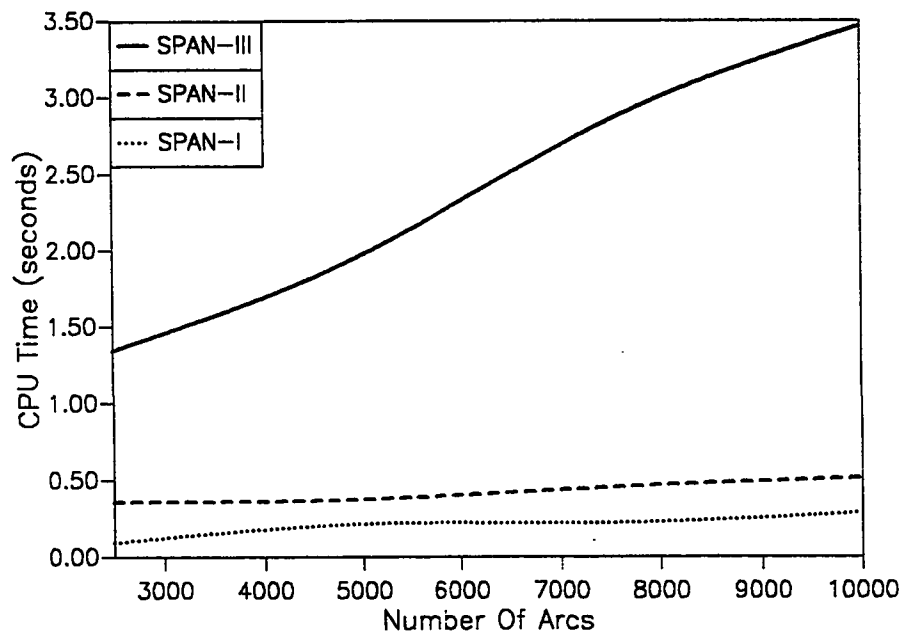


b. Cost Range 1 - 10,000

Figure 3.3 50 x 50 Assignment Problems

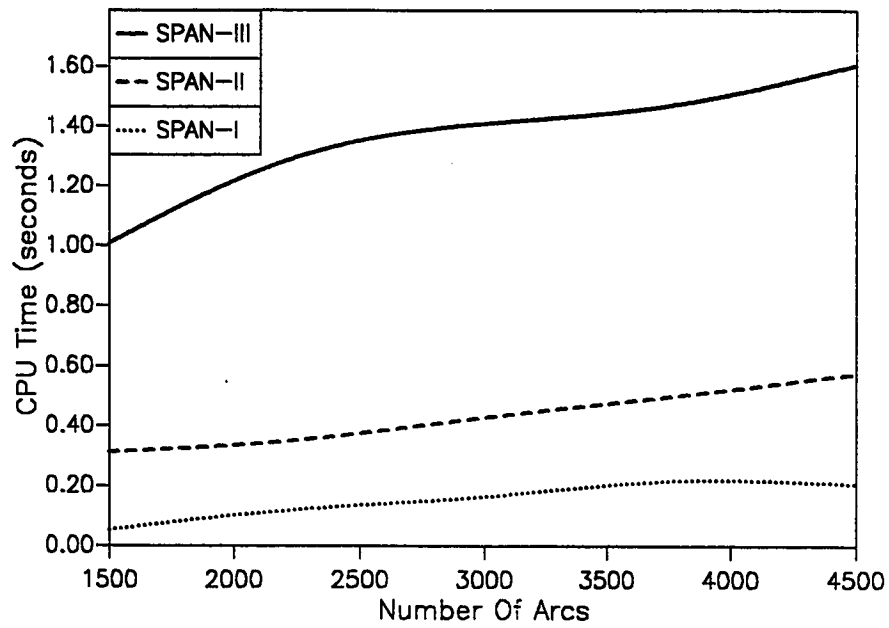


a. Cost Range 1 - 100

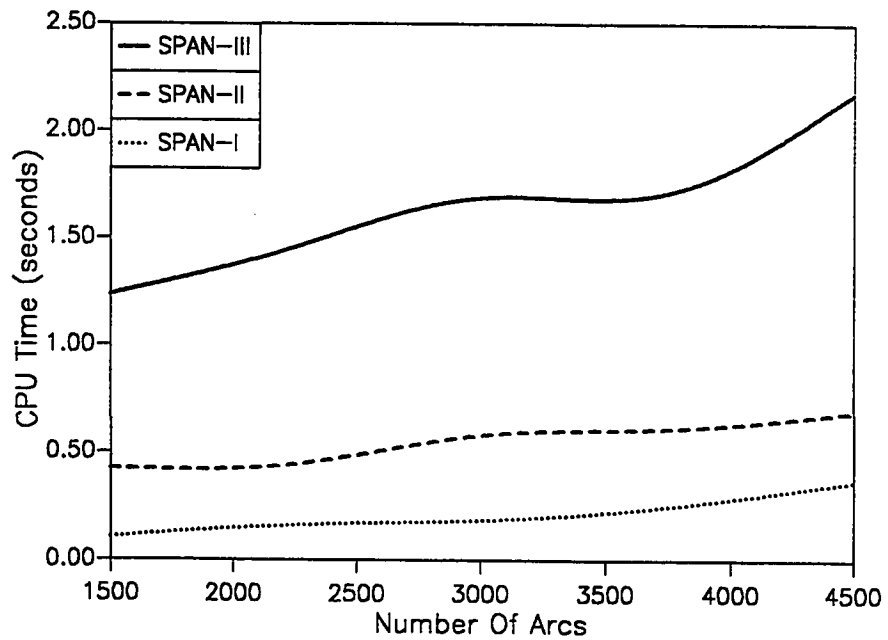


b. Cost Range 1 - 10,000

Figure 3.4 100 × 100 Assignment Problems



a. Cost Range 1 - 100



b. Cost Range 1 - 10,000

Figure 3.5 200 x 200 Assignment Problems

TABLE 3.4 Code Specifications for the SSP Algorithm for the Assignment Problem with Different Shortest Path Algorithms.

Developer	Name	Type	Number of Arrays
1. Duffuaa & Ghassab	SPAN-I	SSP algorithm	$11N + 2A + 2L$
2. Duffuaa & Ghassab	SPAN-II	SSP algorithm	$14N + 3A + 2L$
3. Duffuaa & Ghassab	SPAN-III	SSP algorithm	$11N + 3A$

where,

N = number of nodes

A = number of arcs

L = maximum arc length

Examining Table 3.4, it can be concluded that SPAN-I and SPAN-III have similar memory requirements. Combining the comparison of the codes based on total CPU time and amount of memory required by each code, it can be concluded that the best shortest path algorithm that fits with SSP framework is Dijkstra algorithm.

3.6 Comparative Computational Tests

In this section the best SSP algorithm implemented (SPAN-I) is tested against other efficient algorithms for the assignment problem.

3.6.1 Methods for Comparison

The competing algorithms selected are Carpaneto and Toth code of the Hungarian method [16, 58], a code of the Out-of-Kilter algorithm, part of the SAS system, [32, 63] and Barr *et al.* code of the specialized primal simplex method [9, 36]. The first two codes are referred to as HUNG and ASSIGN, respectively. The third algorithm is known as CAPNET. The four codes tested are written in FORTRAN IV. Four sets of test problems were used. Test problems are generated randomly using NETGEN code with the same random number seed used in [55].

3.6.2 Results and Analysis of the Computation

The cost coefficients for the test problems are randomly generated between 1 and 100 in the first set of problems. For the second set of problems, the cost is randomly generated between 1 and 10,000. The results of the tests are summarized in Tables 3.5 – 3.12. Each table shows the size of the problem, the number of arcs in each problem, cost range and solution times for each problem.

Based on total solution times for the problems shown in Tables 3.5 – 3.12 SPAN-I is about 10 times faster than CAPNET and roughly 16 times faster than HUNG. ASSIGN code could not solve some problems because of memory limitation, hence we did not include it in the calculation.

Graphical representation of solution times of SPAN-I and HUNG versus each problem are shown in Figures 3.6–3.10.

From these results it is discovered that SPAN-I is better than HUNG in case of sparse problems. In the case of 100% dense problems HUNG performed better than SPAN-I. In either case CAPNET has not dominated either SPAN-I or HUNG. We conclude that SPAN-I is about 50 times faster than HUNG in sparse problems, and HUNG is 1.22 times faster than SPAN-I for 100% dense problems.

TABLE 3.5 Solution Times in Seconds for 50×50 Assignment Problems with Cost Range 1-100

Code	Number of Arcs				Total Solution Times	Ratio
	625	1250	1875	2500		
SPAN-I	0.0084	0.0273	0.0293	0.0318	0.0968	1
HUNG	0.1301	0.0859	0.0853	0.0628	0.3641	3.76
ASSIGN	6.49	4.14	3.12	2.67	16.42	169
CAPNET	0.316	0.496	0.6411	0.779	2.2321	23

TABLE 3.6 Solution Times in Seconds for 50×50 Assignment Problems with Cost Range 1-10000

Code	Number of Arcs				Total Solution Times	Ratio
	625	1250	1875	2500		
SPAN-I	0.0338	0.0946	0.1250	0.1796	0.433	1.11
HUNG	0.0779	0.1284	0.0866	0.0954	0.3883	1
ASSIGN	9.27	9.58	9.17	7.89	35.91	92.4
CAPNET	0.356	0.502	0.654	0.780	2.292	5.90

TABLE 3.7 Solution Times in Seconds for 100×100 Assignment Problems with Cost Range 1-100

Code	Number of Arcs				Total Solution Times	Ratio
	2500	5000	7500	10000		
SPAN-I	0.0752	0.1171	0.1972	0.2187	0.6082	1
HUNG	0.2754	0.2451	0.2252	0.2075	0.9532	1.56
ASSIGN	32.74	23.46	16.81	12.91	85.92	141
CAPNET	1.021	1.718	2.250	2.996	7.985	13.1

TABLE 3.8 Solution Times in Seconds for 100×100 Assignment Problems with Cost Range 1-10000

Code	Number of Arcs				Total Solution Times	Ratio
	2500	5000	7500	10000		
SPAN-I	0.0937	0.2132	0.2203	0.2874	0.8146	1
HUNG	0.3925	0.3209	0.2617	0.2202	1.1953	1.46
ASSIGN	73.25	70.48	67.28	61.34	272.35	334
CAPNET	1.031	1.735	2.249	3.007	8.022	9.84

TABLE 3.9 Solution Times in Seconds for 150×150 Assignment Problems with Cost Range 1-100

Code	Number of Arcs					Total Solution Times	Ratio
	1000	3000	5000	7000	9000		
SPAN-I	0.0525	0.0720	0.1055	0.2204	0.2048	0.6552	1
HUNG	2.274	1.831	1.257	0.9433	0.6171	6.9224	10.5
ASSIGN	133.68	116.55	85.18	66.24	69.04	470.69	718
CAPNET	0.671	1.412	2.140	2.657	3.428	10.308	15.7

TABLE 3.10 Solution Times in Seconds for 150×150 Assignment Problems with Cost Range 1-10000

Code	Number of Arcs					Total Solution Times	Ratio
	1000	3000	5000	7000	9000		
SPAN-I	0.1007	0.1852	0.2635	0.2575	0.3861	1.193	1
HUNG	3.589	3.186	2.815	2.4163	2.216	12.0063	10.6
ASSIGN	266.01	288.71	296.72	272.65	CNR	—	—
CAPNET	0.670	1.480	2.151	2.667	3.519	10.487	8.79

CNR = could not run as a result of memory limitation

TABLE 3.11 Solution Times in Seconds for 200×200 Assignment Problems with Cost Range 1-100

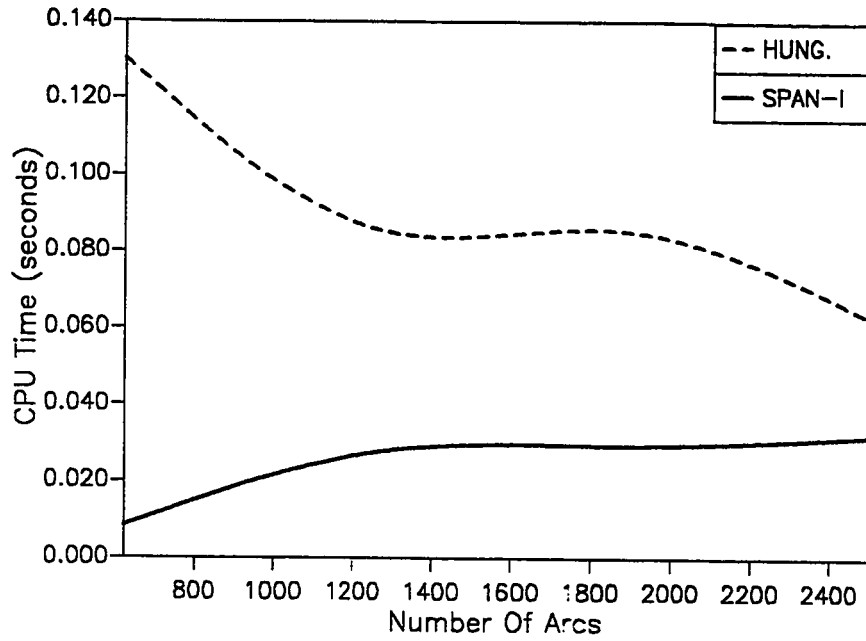
Code	Number of Arcs					Total Solution Times	Ratio
	1500	2250	3000	3750	4500		
SPAN-I	0.0533	0.1201	0.1631	0.2154	0.2054	0.7573	1
HUNG	5.831	5.090	4.578	3.917	2.157	21.573	28.4
ASSIGN	CNR	CNR	CNR	CNR	CNR	—	—
CAPNET	1.043	1.584	1.741	2.006	2.249	8.623	11.4

CNR = could not run as a result of memory limitation

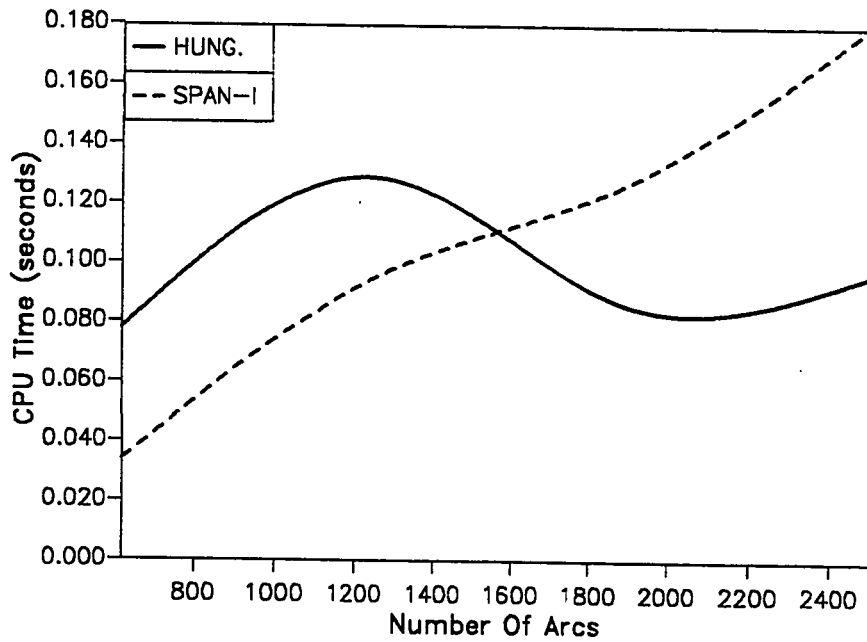
TABLE 3.12 Solution Times in Seconds for 200×200 Assignment Problems with Cost Range 1-10000

Code	Number of Arcs					Total Solution Times	Ratio
	1500	2250	3000	3750	4500		
SPAN-I	0.1058	0.1604	0.1830	0.2495	0.3673	1.066	1
HUNG	13.48	9.907	10.857	8.219	7.003	49.466	46.4
ASSIGN	CNR	CNR	CNR	CNR	CNR	—	—
CAPNET	1.032	1.586	1.739	1.998	2.253	8.608	8.07

CNR = could not run as a result of memory limitation

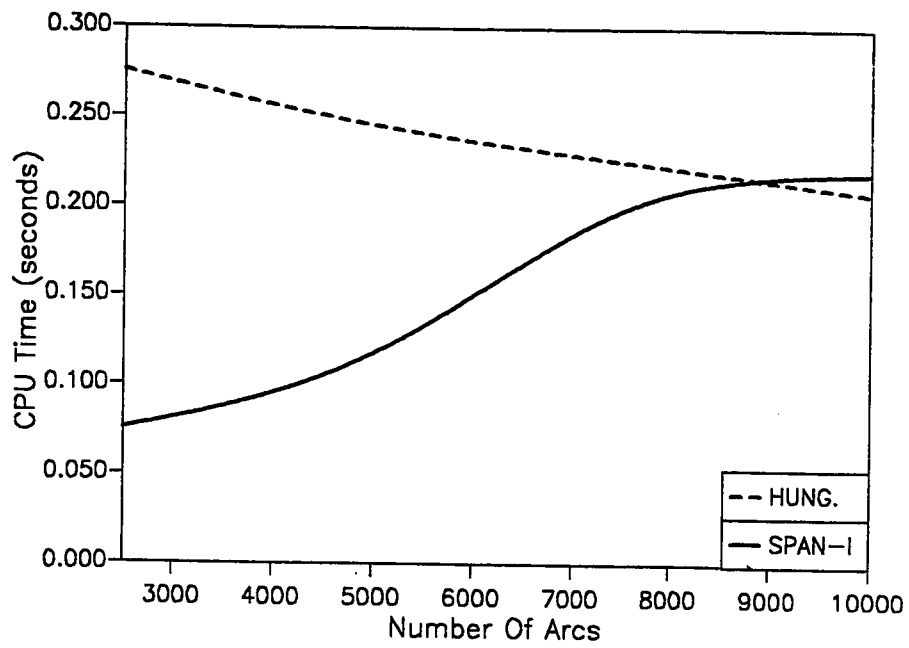


a. Cost Range 1-100

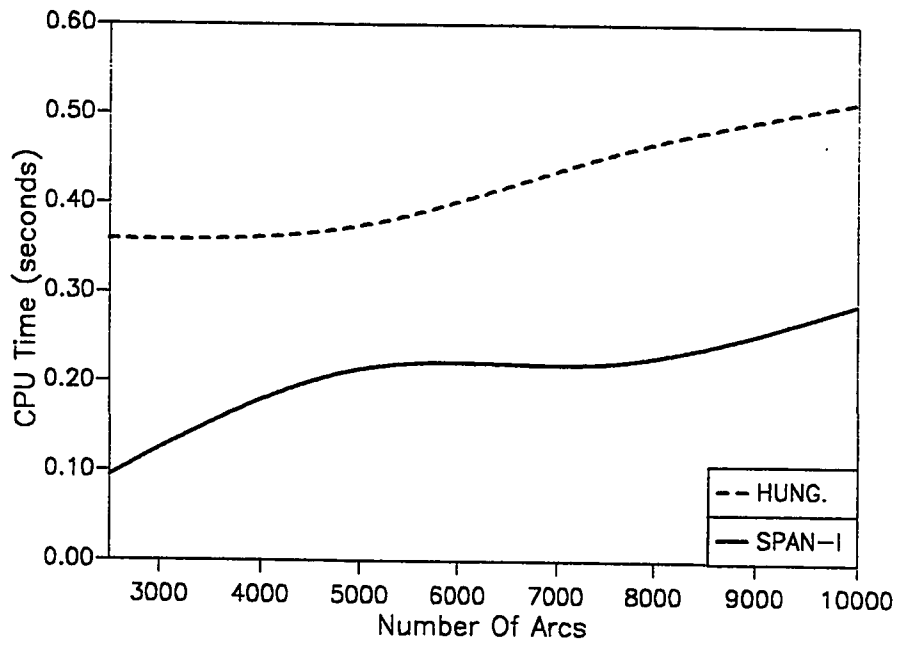


b. Cost Range 1-10,000

Figure 3.6 50 × 50 Assignment Problems

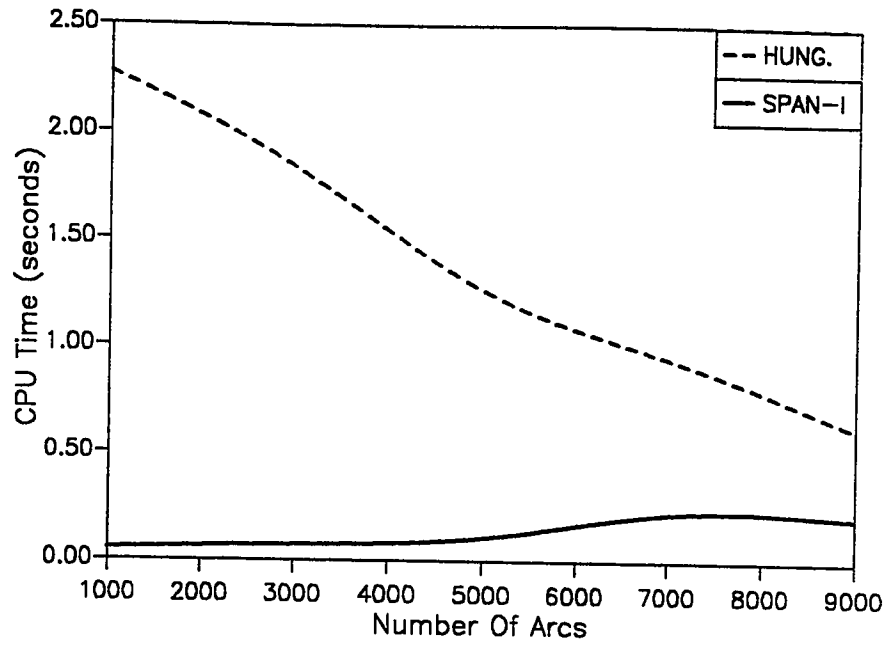


a. Cost Range 1-100

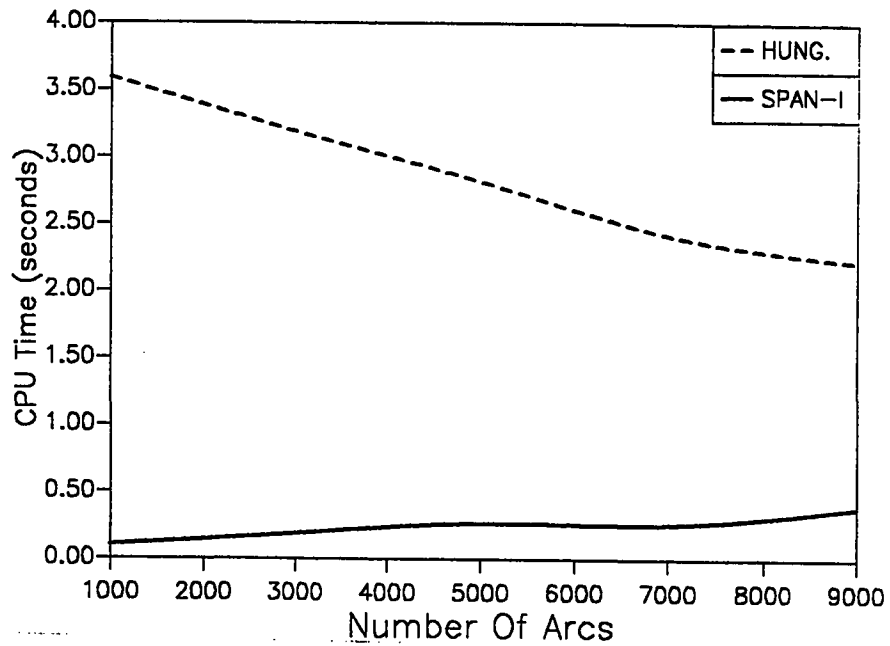


b. Cost Range 1-10,000

Figure 3.7 100 × 100 Assignment Problems

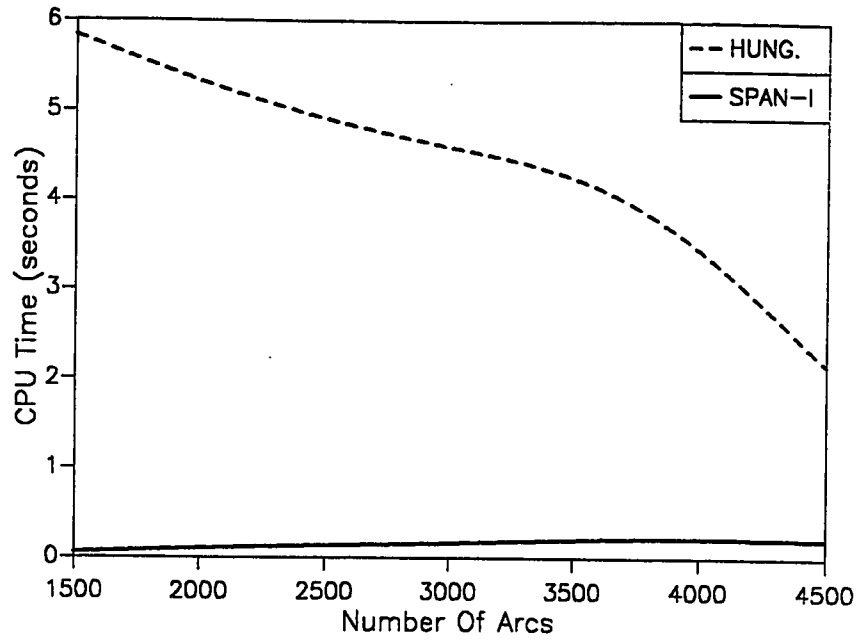


a. Cost Range 1-100

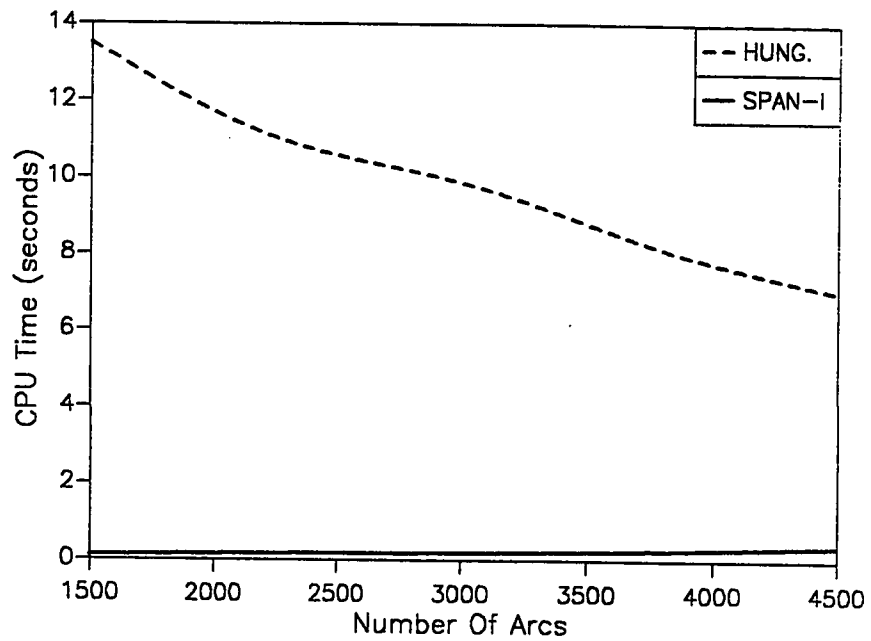


b. Cost Range 1-10,000

Figure 3.8 150 x 150 Assignment Problems



a. Cost Range 1-100



b. Cost Range 1-10,000

Figure 3.9 200 × 200 Assignment Problems

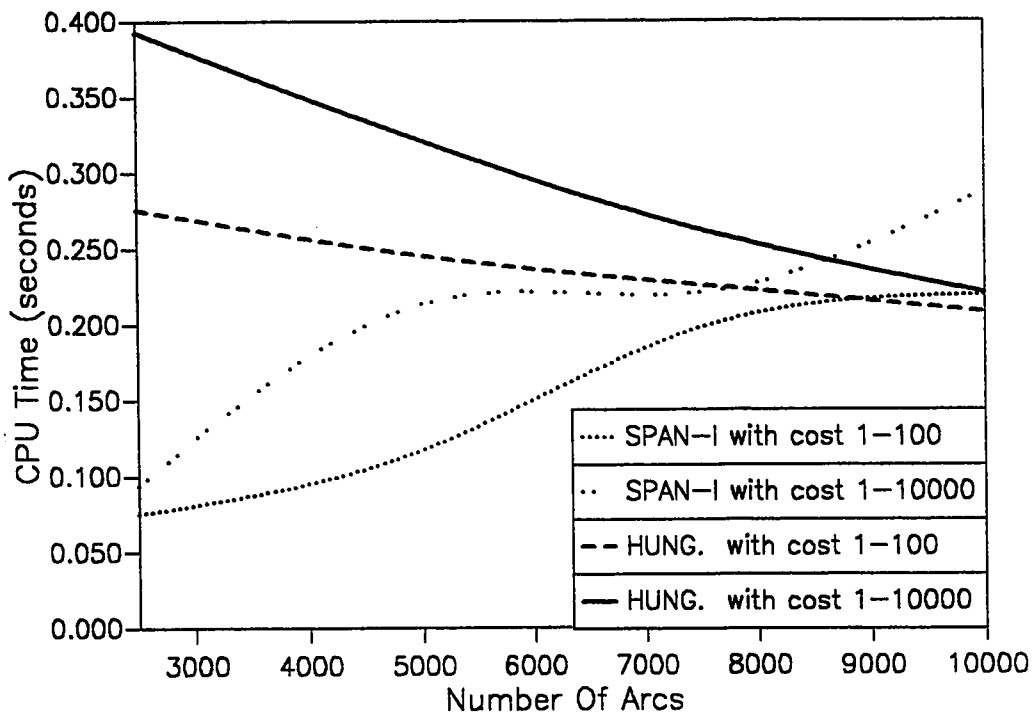


Figure 3.10 100×100 Assignment Problems with Cost Range 1-100 and 1-10,000

In addition to solution-times, algorithms may also be compared on the basis of central memory requirements. Table 3.13 shows the number and size of arrays required by the various codes.

TABLE 3.13 Code Specifications for the Assignment Problem

Developer	Name	Type	Number of Arrays
1. Duffuaa & Ghassab	SPAN-I	SSP algorithm	$11N + 2A + 2L$
2. Carpaneto & Toth	HUNG	Hungarian method	$10N + N^2$
3. Barr <i>et al.</i>	CAPNET	Specialized primal simplex	$6(N + M) + 3A$
4. SAS	ASSIGN	Out-of-Kilter algorithm	Not Available

where,

N = number of origins

M = number of destinations

A = number of arcs

L = maximum arc length

3.7 Conclusion

In this chapter, a successive shortest path algorithm and its implementation is presented. Three codes are developed using Dijkstra, Dantzig and Pape algorithm for solving the shortest path problem in the SSP algorithm. Then, the three codes are tested to find the most efficient shortest path algorithm that fits with the SSP framework. We have verified through computational testing that SPAN-I, which is a code that uses Dijkstra algorithm for solving the shortest path problem, is the most efficient code for solving the assignment problem using a shortest path approach.

SPAN-I is compared with the best known algorithms for solving the assignment problem. Through empirical computational testing it has been verified that SPAN-I is about 50 times faster than HUNG (an implementation of the Hungarian method) on sparse problems, and HUNG is 1.22 times faster than SPAN-I for 100% dense problems. Also, SPAN-I is 10 times faster than CAPNET.

In the next chapter an extension of the SSP algorithm for the assignment problem to solve the semi-assignment problem is presented.

CHAPTER 4

A SUCCESSIVE SHORTEST PATH ALGORITHM FOR THE SEMI-ASSIGNMENT PROBLEM

4.1 Introduction

The semi-assignment problem is a problem of optimally assigning m tasks to n agents such that each task is assigned to exactly one agent but each agent is constrained only by the amount of a resource. It is a network problem whose demand constraints are the same as those of an assignment problem and whose supply constraints are the same as those of a transportation problem. In chapter 3 the SSP algorithm has been implemented and tested for solving the assignment problem. It has been shown that the SSP algorithm solves sparse assignment problems more efficiently than the best known algorithms for solving the assignment problem. In this chapter, the SSP approach is extended for solving the semi-assignment problem. Furthermore, computational testing has been conducted with various algorithms for solving the semi-assignment problem.

The chapter starts by describing the modification needed for the algorithm given in section 3.2 in order to solve the semi-assignment problem. Theoretical results and implementation of the new algorithm are shown in section 4.3. Section 4.4 presents the computer environments used for empirical computational testing. Computational comparisons of different codes against the SSP code are also presented in section 4.4. Section 4.5 concludes the chapter.

4.2 Description of the SSP Algorithm for The Semi-Assignment Problem

An $m \times n$ semi-assignment problem is defined as:

$$\begin{aligned}
 &\text{Minimize} && \sum_{(i,j) \in E} c_{ij} x_{ij} \\
 &\text{Subject to} && \\
 &&& \sum_{(i,j) \in FS(i)} x_{ij} = b_i, i \in I = \{1, 2, 3, \dots, m\} \quad (4.1) \\
 &&& \sum_{(i,j) \in RS(j)} x_{ij} = 1, j \in I = \{1, 2, 3, \dots, n\} \\
 &&& x_{ij} \geq 0, (i, j) \in E
 \end{aligned}$$

Given the semi-assignment problem, we say $X = (x_{ij})$ is a *tentative solution* provided \exists at least one $j \in J$ such that

$$\sum_{(i,j) \in FS(i)} x_{ij} = b_i, \quad i \in I. \quad (4.2)$$

Since the semi-assignment network will remain fixed in the following discussion, we denote the semi-assignment problem equipped with a tentative solution X by (m, n, C, b, X) .

We say that (m, n, C, b, X) is in the standard form if $c_{ij} \geq 0$ for $(i, j) \in E$ and $c_{ij} = 0$ if $x_{ij} \geq 0$.

In the starting procedure for SSP a tentative solution is defined as follows.

First,

$$\hat{c}_i = \min_{(i,p) \in FS(i)} \{c_{ip}\} \quad (4.3)$$

must be determined for $i \in I$. $x_{ij} = b_i$ for some j such that $c_{ij} = \hat{c}_i$. For this X , (m, n, C, b, X) may not be in the standard form. However, the forward start of i may be scaled by setting

$$c_{ip} \leftarrow c_{ip} - \hat{c}_i, \quad \text{for } (i, p) \in FS(i). \quad (4.4)$$

The resulting (m, n, C, b, X) is in the standard form.

For a given tentative solution X , we let

$$a_j = \sum_{(i,j) \in RS(j)} x_{ij} \quad (4.5)$$

The modified semi-assignment problem relative to (m, n, C, b, X) is defined as follows:

$$\text{Minimize } \sum_{(i,j) \in E} c_{ij} x_{ij}$$

Subject to

$$\sum_{(i,j) \in FS(i)} x_{ij} = b_i, \quad i \in I \quad (4.6)$$

$$\sum_{(i,j) \in RS(i)} x_{ij} = a_j, \quad j \in I$$

$$x_{ij} \geq 0, \quad (i, j) \in E$$

We note that when (m, n, C, b, X) is in standard form, X provides an optimal solution to the modified semi-assignment problem in 4.6 relative to (m, n, C, b, X) .

A destination node j is said to be *abundant* relative to X when $a_j > 1$. Likewise, j is said to be *deficient* relative to X when $a_j = 0$.

The method introduced in section 3.2 for solving the assignment problem is the same for solving semi-assignment problem. However, there are changes in step 2 in constructing and solving the $SP(C^k, X^k, d^k)$ for the semi-assignment case. The following are the major changes involved in constructing and solving the shortest path problem for the semi-assignment case:

1. We refer to $(i \rightarrow A_i)$ as the i -th shortest path node in the case of the assignment problem and we have n such nodes. In the case of the semi-assignment problem, we may have a source i supplying several destinations. Let the number of destinations supplied by node i (i.e. $x_{ij} > 0$) be r_i (e.g., $(i \rightarrow X_{i,1}), (i \rightarrow X_{i,2}), \dots, (i \rightarrow X_{i,r_i})$). This implies in the shortest path problem $SP(C^k, X^k, d^k)$, two nodes may have the same origin i , for example $(i \rightarrow X_{i,1})$ and $(i \rightarrow X_{i,2})$. Therefore, in solving the $SP(C^k, X^k, d^k)$ a tie may occur when scanning the predecessor of these two nodes. The tie is broken arbitrarily. This does not occur in the case of the assignment problem, since every source of the assignment problem supplies only one destination.
2. In step 5 of SSP in section 3.2, the assignments from the abundant node are reversed in the following manner:

Step 5 (modified)

Whenever the i -th shortest path node is in $S(v^k)$, $i \neq m + 1$, set $X_{i,0}^{k+1} = X_{\ell,Q}^k$, where O is the destination assignment number for node i , Q is the destination assignment number of node ℓ , and $\ell = P_i^k$. For the abundant node $X_{i,r_i+1}^{k+1} = X_{\ell,Q}^k$, if $X_{i,0}^k$ receives more than one unit from node i (i.e., $x_{iX_{i,0}^k} > 1$). For all other origins i , $X_{i,0}^{k+1} = X_{i,0}^k$. Set $k \rightarrow k + 1$ and go to step 1 of the SSP algorithm.

In order to fix ideas, we present an example showing how $SP(C, X, d)$ is defined in a particular case. Let the semi-assignment problem be as shown in Figure 4.1, where arc (i, j) is labelled with cost c_{ij} .

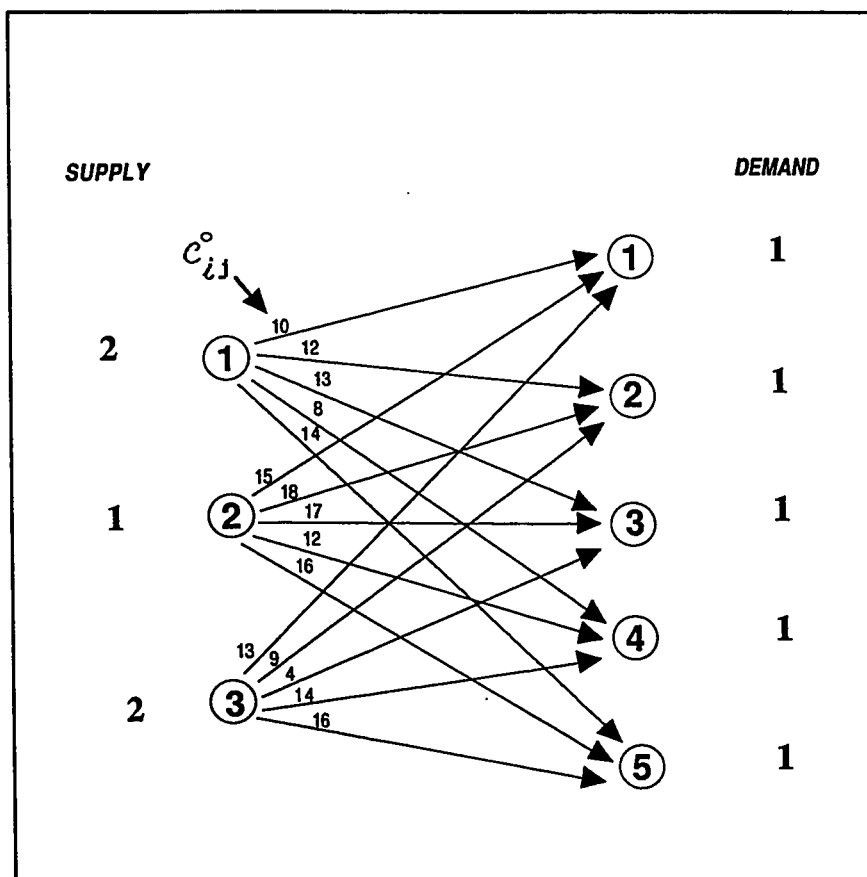


Figure 4.1 An Example of a Semi-assignment Problem.

Step 0

Put the problem into standard form:

$$\hat{c}_1 = 8, \quad \hat{c}_2 = 12, \quad \hat{c}_3 = 4,$$

$$c_{11}^1 = c_{11}^0 - \hat{c}_1 = 10 - 8 = 2, \quad c_{21}^1 = c_{21}^0 - \hat{c}_2 = 15 - 12 = 3$$

$$c_{12}^1 = c_{12}^0 - \hat{c}_1 = 12 - 8 = 4, \quad c_{22}^1 = c_{22}^0 - \hat{c}_2 = 18 - 12 = 6$$

$$c_{13}^1 = c_{13}^0 - \hat{c}_1 = 13 - 8 = 5, \quad c_{23}^1 = c_{23}^0 - \hat{c}_2 = 17 - 12 = 5$$

$$c_{14}^1 = c_{14}^0 - \hat{c}_1 = 8 - 8 = 0, \quad c_{24}^1 = c_{24}^0 - \hat{c}_2 = 12 - 12 = 0$$

$$c_{15}^1 = c_{15}^0 - \hat{c}_1 = 14 - 8 = 6$$

$$c_{25}^1 = c_{25}^0 - \hat{c}_2 = 16 - 12 = 4$$

$$c_{31}^1 = c_{31}^0 - \hat{c}_3 = 13 - 4 = 9, \quad c_{32}^1 = c_{32}^0 - \hat{c}_3 = 9 - 4 = 5$$

$$c_{33}^1 = c_{33}^0 - \hat{c}_3 = 4 - 4 = 0, \quad c_{34}^1 = c_{34}^0 - \hat{c}_3 = 14 - 4 = 10$$

$$c_{35}^1 = c_{35}^0 - \hat{c}_3 = 16 - 4 = 12$$

Therefore,

$$X^1 = (X_{11}^1, X_{21}^1, X_{31}^1) = (4, 4, 3) \text{ and}$$

$$a_1^1 = 0, \quad a_2^1 = 0, \quad a_3^1 = 2, \quad a_4^1 = 3, \quad a_5^1 = 0$$

$$k = 1.$$

Iteration 1.

Step 1: Choose a deficient node, $d^1 = 1$

Step 2: Solve $SP(C^1, X^1, d^1)$.

The network for $S(C^1, X^1, d^1)$ is shown in Figure 4.2 where a shortest path node $(i \rightarrow X_{i,0})$ is shown as a node with an upper label (i) and a lower label $(X_{i,0})$. The arcs of the shortest path network are labelled with their lengths.

The results of this step are:

$$D_1^1 = 2, \quad P_1^1 = 4, \quad v^1 = \begin{pmatrix} 1 \\ 4 \end{pmatrix} \quad \text{and} \quad L^1 = 2$$

Step 3:

$$R_1^1 = D_1^1 - L^1 = 2 - 2 = 0, \quad K_4^1 = L^1 - D_1^1 = 2 - 2 = 0$$

$$R_2^1 = R_3^1 = K_1^1 = K_2^1 = K_3^1 = K_5^1 = 0$$

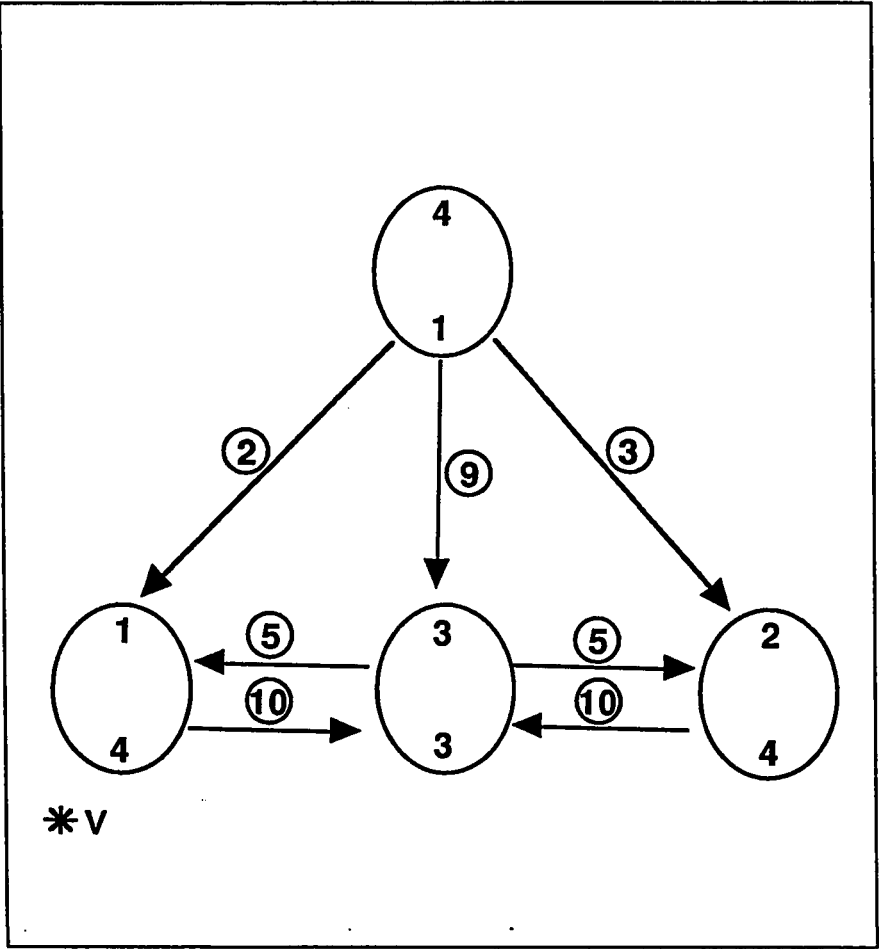


Figure 4.2 An Example of $SP(C, X, d)$

Step 4

$$c_{11}^2 = c_{11}^1 - R_1^1 - K_1^1 = 2 - 0 - 0 = 2, \quad c_{13}^2 = c_{13}^1 - R_1^1 - K_3^1 = 5 - 0 - 0 = 5$$

$$c_{12}^2 = c_{12}^1 - R_1^1 - K_2^1 = 4 - 0 - 0 = 4, \quad c_{14}^2 = c_{14}^1 - R_1^1 - K_4^1 = 0 - 0 - 0 = 0$$

$$c_{15}^2 = c_{15}^1 - R_1^1 - K_5^1 = 6 - 0 - 0 = 6$$

$$c_{21}^2 = c_{21}^1 - R_2^1 - K_1^1 = 3 - 0 - 0 = 3, \quad c_{23}^2 = c_{23}^1 - R_2^1 - K_3^1 = 5 - 0 - 0 = 5$$

$$c_{22}^2 = c_{22}^1 - R_2^1 - K_2^1 = 6 - 0 - 0 = 6, \quad c_{24}^2 = c_{24}^1 - R_2^1 - K_4^1 = 0 - 0 - 0 = 0$$

$$c_{25}^2 = c_{25}^1 - R_2^1 - K_5^1 = 4 - 0 - 0 = 4$$

$$c_{31}^2 = c_{31}^1 - R_3^1 - K_1^1 = 9 - 0 - 0 = 9, \quad c_{33}^2 = c_{33}^1 - R_3^1 - K_3^1 = 0 - 0 - 0 = 0$$

$$c_{32}^2 = c_{32}^1 - R_3^1 - K_2^1 = 5 - 0 - 0 = 5, \quad c_{34}^2 = c_{34}^1 - R_3^1 - K_4^1 = 10 - 0 - 0 = 10$$

$$c_{35}^2 = c_{35}^1 - R_3^1 - K_5^1 = 12 - 0 - 0 = 12$$

Step 5

$$X^2 = (X_{11}^2, X_{12}^2, X_{21}^2, X_{31}^2) = (4, 1, 4, 3)$$

$$a_1^2 = 1, \quad a_2^2 = 0, \quad a_3^2 = 2, \quad a_4^2 = 2, \quad a_5^2 = 0$$

let $k = 2$, go to step 1

Iteration 2.

Step 1 Choose a deficient node, $d^2 = 2$.

Step 2 Solve $SP(C^2, X^2, d^2)$

The results of this step are:

$$D_1^2 = 4, \quad P_1^2 = 4, \quad v^2 = \begin{pmatrix} 1 \\ 4 \end{pmatrix}, \quad \text{and} \quad L^2 = 4$$

After solving the shortest path problem the following nodes are permanently labelled:

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} 1 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

Step 3

$$R_1^2 = D_1^2 - L^2 = 4 - 4 = 0$$

$$K_1^2 = L^2 - D_1^2 = 4 - 4 = 0$$

$$K_4^2 = L^2 - D_1^2 = 4 - 4 = 0$$

$$R_2^2 = R_3^2 = K_2^2 = K_3^2 = K_5^2 = 0$$

Step 4

$$c_{11}^3 = c_{11}^2 - R_1^2 - K_1^2 = 2 - 0 - 0 = 2$$

$$c_{12}^3 = 4, \quad c_{13}^3 = 5, \quad c_{14}^3 = 0, \quad c_{15}^3 = 6$$

$$c_{21}^3 = c_{21}^2 - R_2^2 - K_1^2 = 3 - 0 - 0 = 3$$

$$c_{22}^3 = 6, \quad c_{23}^3 = 5, \quad c_{24}^3 = 0, \quad c_{25}^3 = 4$$

$$c_{31}^3 = C_{31}^2 - R_3^2 - K_1^2 = 9 - 0 - 0 = 9$$

$$c_{32}^3 = 5, \quad c_{33}^3 = 0, \quad c_{34}^3 = 10, \quad c_{35}^3 = 12$$

Step 5

$$X^3 = (X_{11}^3, X_{12}^3, X_{21}^3, X_{31}^3) = (2, 1, 4, 3)$$

$$a_1^3 = 0, \quad a_2^3 = 1, \quad a_3^3 = 2, \quad a_4^3 = 1, \quad a_5^3 = 0$$

let $k = 3$, go to step 1

Iteration 3.

Step 1 Choose a deficient node, $d^3 = 5$

Step 2 Solve $SP(C^3, X^3, d^3)$

The results of this step are:

$$D_1^3 = 4, \quad D_2^3 = 4, \quad D_3^3 = 9,$$

$$P_1^3 = 2, \quad P_2^3 = 0, \quad P_3^3 = 1$$

$$v^3 = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \quad \text{and} \quad L^3 = 9$$

Step 3

$$R_1^3 = D_1^3 - L^3 = 4 - 9 = -5, \quad R_2^3 = D_2^3 - L^3 = 4 - 9 = -5, \quad R_3^3 = D_3^3 - L^3 = 9 - 9 = 0$$

$$K_2^3 = L^3 - D_1^3 = 9 - 4 = 5, \quad K_3^3 = L^3 - D_3^3 = 9 - 9 = 0, \quad K_4^3 = L^3 - D_2^3 = 9 - 4 = 5$$

$$K_1^3 = L^3 - D_1^3 = 9 - 4 = 5, \quad K_5^3 = 0$$

Step 4

$$c_{11}^4 = c_{11}^3 - R_1^3 - K_1^3 = 2, \quad c_{12}^4 = 4 - (-5) - 5 = 4, \quad c_{13}^4 = 5 - (-5) - 0 = 10$$

$$c_{14}^4 = 0 - (-5) - 5 = 0, \quad c_{15}^4 = 6 - (-5) - 5 = 11$$

$$c_{21}^4 = c_{23}^3 - R_2^3 - K_1^3 = 3, \quad c_{22}^4 = 6 - (-5) - 5 = 6, \quad c_{23}^4 = 5 - (-5) - 0 = 10$$

$$c_{24}^4 = 0 - (-5) - 5 = 0, \quad c_{25}^4 = 4 - (-5) - 0 = 9$$

$$c_{31}^4 = c_{31}^3 - R_3^3 - K_1^3 = 4, \quad c_{32}^4 = 5 - 0 - 5 = 0, \quad c_{33}^4 = 0 - 0 - 0 = 0$$

$$c_{34}^4 = 10 - 0 - 5 = 5, \quad c_{35}^4 = 12 - 0 - 0 = 12$$

Step 5

$$X^4 = (X_{11}^4, X_{12}^4, X_{21}^4, X_{31}^4, X_{32}^4) \quad (4, 1, 5, 3, 2)$$

$$a_1^4 = 1, \quad a_2^4 = 1, \quad a_3^4 = 1, \quad a_4^4 = 1, \quad a_5^4 = 1$$

Let $k = 4$, go to step 1

Iteration 4.

Step 1 No deficient node exists. Stop.

Therefore, optimal assignments are:

$$(1 \rightarrow 1), (1 \rightarrow 4), (2 \rightarrow 5), (3 \rightarrow 2), (3 \rightarrow 3)$$

$$\text{with total cost} = 10 + 8 + 16 + 9 + 4 = 47.$$

4.3 Theoretical Results and Implementation

In this section certain theoretical properties of the SSP algorithm for the semi-assignment problem are examined. Theorem 1 is a convergence result, while theorem 2 deals with the computational complexity of the algorithm. Theorem 3 provides a verification of the infeasibility criterion in step 2 of the algorithm. Implementation of the algorithm is discussed in subsection 4.3.2

4.3.1 Theoretical Results

Theorem 1 *If SSP does not stop because of the infeasibility test in step 2, it reaches optimality in at most $n - 1$ iterations.*

Proof: We proceed by induction. We have (m, n, C^1, b, X^1) is in standard form with at most $n - 1$, deficient destinations. If we assume that (m, n, C^k, b, X^k) is in standard form with $q \geq 1$, deficient destinations, it follows that $(m, n, C^{k+1}, b, X^{k+1})$ is in standard form with $q - 1$ deficient destinations. At each iteration the number of deficient nodes is reduced by one. Therefore the theorem follows.

Theorem 2. *SSP algorithm has a $O(n^3)$ computational bound.*

Proof: When the algorithm does not indicate an infeasible semi-assignment problem, it requires at most $n - 1$ iterations by Theorem 1. This result, together with the $O(n^2)$ computational bound, where n is the number of nodes in the shortest path tree, for the Dijkstra shortest path algorithm, implies the $O(n^3)$ bound in

this case.

Theorem 3. *The original semi-assignment problem is infeasible if and only if the shortest path algorithm fails to label an abundant node of $SP(C^k, X^k, d^k)$ on some iteration k .*

Proof: Half the proof follows from Theorem 1. We proceed with the remaining half. Suppose that the algorithm fails to label an abundant node. Let N_1 be the set of origins i such that $(i \rightarrow q)$ is permanently labelled for some q , and let N_2 be the set of destinations j such that $(p \rightarrow j)$ is permanently labelled for some p . All arcs of the original semi-assignment network which terminate in N_2 must originate in N_1 by the way $SP(C^k, X^k, d^k)$ is defined. Since N_2 contains one more element than N_1 (namely, d^k) it is clear that the original semi-assignment problem is infeasible.

4.3.2 Implementation of the SSP Algorithm for the Semi-Assignment Problem

As in any computer code, there are still minor coding changes which could be performed to enhance overall efficiency. Therefore, our goal has been to make the code as efficient as possible.

An implementation of this algorithm is the same implementation discussed in chapter 3 for solving the assignment problem. However, the implementation techniques for creating and solving the shortest path algorithm (section 3.3.2.1)

needs some modifications.

The assignment of each source node i to a destination node j is stored in an array called X (initially $X := \phi$) of length m . Each time a destination node j is satisfied from a source node i , it is linked to the list in position i of the array X .

When reversing the assignments in step 5, a destination node in $S(v^k)$ may change its source node. If the abundant node in the modified semi-assignment problem receiving only one unit from a source node i in $S(v^k)$, then the position of this abundant node is occupied by the destination node of the predecessor of node i in $S(v^k)$. If the abundant node j in the modified semi-assignment problem receiving more than one unit from a source node i in $S(v^k)$, the destination of the predecessor of node i in $S(v^k)$ is supplied from node i and the abundant destination node j is supplied also from node i . For the rest of the path $S(v^k)$ continue by successively examining the predecessors of v until the root node is encountered.
previous.

4.4 Comparative Computation Tests

4.4.1 Experimental Design

An in-core FORTRAN code called SPSN, has been developed for the semi-assignment problem utilizing the SSP algorithm, the computational considerations presented in section 3.4.1 are used in SPSN. The computer environment introduced in section 3.4.2 is the same environment we used to test SPSN. Also, the problems used in the tests were randomly generated using NETGEN. In contrast to the assignment test problems, the specification of the semi-assignment test problems vary greatly in both the number of nodes and arcs. The test problems vary in size from 50 origins and 500 destinations to 400 origins and 4000 destinations. The number of arcs varies from 2000 arcs to 16,000 arcs. The cost data were randomly generated between 1 - 1000 for the first test problems. In the second test problems the cost data were randomly generated between 1 - 10,000. Solution times are an average of five runs for each problem.

4.4.2 Computational Comparison

The codes which are used for comparison are CAPNET a code of specialized primal simplex method [9, 36], TRANS, code of the Out-of-Kilter algorithm part of SAS system [32, 65], and the third code is SPSAN. SPSAN is developed in this

thesis to solve the semi-assignment problem as an assignment problem using the SSP algorithm. The code originally is SPAN-I and we did some modification to it so that it could convert the semi-assignment problem to assignment problem. All of these codes are in-core codes and they are all coded in FORTRAN.

In order to compare the codes against SPSN, we solved the same semi-assignment problems used by Barr *et al.* [6]. The results of the comparison are shown in Tables 4.1 and 4.2. Table 4.1 shows the problem size, the density (in terms of the number of arcs), and solution times. Table 4.2 presents the same problems in Table 4.1 with cost range between 1 - 10,000. Upon examining Tables 4.1 and 4.2, the results indicate that SPSN is the fastest method for solving the semi-assignment problems in comparison with the tested codes. The SPSN times strictly dominate the times of the other codes. Comparing the sum of the total solution times, the SPSN code is 1.3 times faster than CAPNET and 1.8 times faster than SPSAN. TRANS could not solve any problem because the size of the problems is too large.

TABLE 4.1 Solution Times in Seconds on Semi-assignment
Problems with a cost range of 1-1000.

No. of Nodes		Times (in seconds)			
$m \times n$	No. of Arcs	CAPNET	TRANS	SPSAN	SPSN
50 × 500	2,000	1.247	CNR	2.179	0.6332
50 × 500	5,000	2.079	CNR	2.827	0.7456
50 × 500	10,000	3.740	CNR	4.158	1.568
50 × 1000	4,000	2.515	CNR	3.206	1.292
50 × 1000	10,000	5.384	CNR	5.820	2.904
50 × 1000	20,000	7.830	CNR	8.105	4.502
100 × 1000	4,000	3.064	CNR	3.763	1.742
100 × 1000	10,000	5.235	CNR	6.022	3.625
100 × 1000	16,000	7.930	CNR	8.291	4.370
400 × 4000	10,000	19.56	CNR	22.12	13.15
400 × 4000	16,000	25.13	CNR	28.36	17.95
Total Solution Times		83.714	—	94.36	52.4818
Ratio		1.59	—	1.79	1

CNR = could not run as a result of memory limitations.

TABLE 4.2. Solution Times in Seconds on Semi-assignment Problems with a cost range of 1-10,000.

No. of Nodes		Times (in seconds)		
$m \times n$	No. of Arcs	CAPNET	SPSAN	SPSN
50 × 500	2,000	1.483	3.680	1.113
50 × 500	5,000	2.563	4.529	1.498
50 × 500	10,000	4.112	7.664	2.448
50 × 1000	4,000	2.752	5.725	2.059
50 × 1000	10,000	5.692	10.26	4.122
50 × 1000	20,000	8.042	14.85	6.751
100 × 1000	4,000	3.137	5.057	2.015
100 × 1000	10,000	5.652	9.475	6.074
100 × 1000	16,000	8.195	15.19	7.969
400 × 4000	10,000	21.30	34.80	21.14
400 × 4000	16,000	27.45	45.21	26.30
Total Solution Times		90.378	156.44	81.489
Ratio		1.10	1.91	1

4.4.3 Memory Requirements of the Codes

Table 4.3 indicates the number of origins, destinations, and arc length arrays required for each of the codes tested for solving semi-assignment problems except for the TRANS code. The storage requirements of this code were not available. It should be noted that memory requirements of SPSN and CAPNET are quite close. Further, it is important to note that based on the tested problems CAPNET requires the least amount of memory of all the codes; this feature gives CAPNET the advantage of solving large problems other codes can not solve.

TABLE 4.3 Code Specifications for the Semi-assignment Problem

Developer	Name	Type	Number of arrays
1. Duffuaa and Ghassab	SPSN	SSP algorithm	$9M + 3N + 4A + 2L$
2. Duffuaa and Ghassab	SPSAN	SSP algorithm	$11N + 2A + 2L$
3. SAS	TRANS	Out-of-Kilter method	Not available
4. Barr <i>et al.</i>	CAPNET	Specialized Primal Simplex algorithm	$6M + 6N + 3A$

where,

M = number of origins

N = number of destinations

A = number of arcs

L = maximum arc length

4.5 Conclusion

The new algorithm presented in this chapter is a modification of the SSP algorithm for the assignment problem in order to solve the semi-assignment problem. Coding this algorithm and comparing it with available codes for the semi-assignment problem has indicated that the SSP algorithm is a very efficient approach for the semi-assignment problem. The results of the comparison have shown that SPSN is 1.3 times faster than CAPNET and 1.8 times faster than SPSAN. The code TRANS could not solve any problem because of memory limitations. Based on memory requirements, no single code seems to dominate the others; it depends on the size of M , N , A and L .

In the following chapter the SSP approach is generalized to solve the transportation problem.

CHAPTER 5

A SUCCESSIVE SHORTEST PATH ALGORITHM FOR THE TRANSPORTATION PROBLEM

5.1 Introduction

The transportation problem is a type of network flow problem. Each source node is able to supply a specified number of units; each destination node has a demand for a specified number of units. A cost is incurred for shipping units along arcs between sources and destination. The objective is to satisfy demands at minimum total cost. Some arcs may have capacities and lower flow bounds (capacitated transportation problem), this case is not covered in the thesis. Several algorithms have been developed for solving uncapacitated transportation problems, namely [6, 11, 17]. The SSP algorithm implemented and tested in Chapter 3 for solving the assignment problem, has been extended in Chapter 4 to solve the semi-assignment problem. The purpose of this chapter is to generalize the SSP algorithm to solve the transportation problem. Furthermore, to compare the performance of the generalized SSP algorithm for the transportation problem with CAPNET and TRANS. TRANS is an implementation of Out-of-Kilter algorithm for the transportation problem and is a part of the SAS system [64].

This chapter begins by describing the SSP algorithm for the transportation problem. Theoretical results and the implementation of the algorithm are presented in Section 5.3. Section 5.4 reviews the computer environment for the computational study. Results of the performance of the SSP algorithm for the transportation problem as compared to other algorithms is shown in Section 5.4. Section 5.5 concludes the chapter.

5.2 Description of the SSP Algorithm for The Transportation Problem

Consider one commodity in one time period. The commodity is available in given quantities a_i at m origins and required in given quantities b_j at n destinations. The total amount available equals the total amount required. The unit cost of routing the commodity from origin i to destination j is denoted by c_{ij} . The problem is to determine the number of units x_{ij} to be routed from each origin to each destination to minimize the total cost. The problem may be rewritten in linear programming form as follows:

$$\begin{aligned}
 &\text{Minimize} && \sum_{(i,j) \in E} c_{ij}x_{ij} \\
 &\text{Subject to} && \\
 &&& \sum_{(i,j) \in FS(i)} x_{ij} = a_i, \quad i \in I = \{1, 2, 3, \dots, m\} \\
 &&& \sum_{(i,j) \in RS(j)} x_{ij} = b_j, \quad j \in J = \{1, 2, 3, \dots, n\} \\
 &&& x_{ij} \geq 0, \quad (i, j) \in E
 \end{aligned} \tag{5.1}$$

for a balanced transportation problem $\sum_i a_i = \sum_j b_j$.

Given the transportation problem, we say $X = (x_{ij})$ is a *tentative solution* provided there exists at least one $j \in J$, such that

$$\sum_{(i,j) \in FS(i)} x_{ij} = a_i, \quad i \in I. \tag{5.2}$$

Since the transportation problem will remain fixed in the following discussion, we denote the transportation problem equipped with a tentative solution X by

(m, n, C, a, b, X) .

We say that (m, n, C, a, b, X) is in the standard form if $c_{ij} \geq 0$ for $(i, j) \in E$ and $c_{ij} = 0$ if $x_{ij} \geq 0$.

In the starting procedure for SSP a tentative solution is defined as follows. First,

$$\hat{c}_i = \min_{(i,p) \in FS(i)} \{c_{ip}\} \quad (5.3)$$

must be determined for $i \in I$. $x_{ij} = a_i$ for some j such that $c_{ij} = \hat{c}_i$. For this X , (m, n, C, a, b, X) may not be in the standard form. However, the forward star of i may be scaled by setting

$$c_{ip} \leftarrow c_{ip} - \hat{c}_i, \quad \text{for } (i, p) \in FS(i). \quad (5.4)$$

The resulting (m, n, C, a, b, X) is in the standard form.

For a given tentative solution X , we let $b'_j = \sum_{(i,j) \in RS(j)} x_{ij}$. The modified transportation problem relative to (m, n, C, a, b, X) is defined as follows:

$$\begin{aligned} &\text{Minimize} && \sum_{(i,j) \in E} c_{ij} x_{ij} \\ &\text{Subject to} && \\ &&& \sum_{(i,j) \in FS(i)} x_{ij} = a_i, \quad i \in I \\ &&& \sum_{(i,j) \in RS(j)} x_{ij} = b'_j, \quad j \in J \\ &&& x_{ij} \geq 0, \quad (i, j) \in E \end{aligned} \quad (5.5)$$

We note that when (m, n, C, a, b, X) is in standard form, X provides an optimal solution to the modified transportation problem in (5.5), relative to (m, n, C, a, b, X) .

A destination node j is said to be *abundant* relative to X when $b_j > b_j$. If $b_j < b_j$, the destination node is said to be *deficient* relative to X . A destination node j is said to be *neutral* relative to X when $b_j = b_j$.

Suppose (m, n, C, a, b, X) is in standard form and d is some deficient node with respect to X . The shortest path problem relative to (m, n, C, a, b, X) and d is denoted $SP(C, X, d)$ and is defined as in the case of semi-assignment problem explained in Section 4.2. However, one difference exists between the $SP(C, X, d)$ for the semi-assignment and the transportation problems, that in the case of the transportation deficient nodes with positive shipment can be a part of the $SP(C, X, d)$.

The major steps of SSP for the transportation problem are as follows:

0. Define X^1 and transform C^0 to C^1 by scaling as described above so that (m, n, C^1, a, b, X^1) is in standard form. Set $k = 1$.
1. Choose a destination node d^k which is deficient relative to X^k . If no deficient node exists, stop, X^k defines an optimal solution.
2. Solve $SP(C^k, X^k, d^k)$. The shortest algorithm is terminated as soon as an abundant shortest path node is permanently labelled. If the shortest path algorithm fails to permanently label an abundant node, stop, the transportation problem is infeasible. Otherwise, the results of this step are D^k, P^k, v^k and L^k .
3. For each permanently labelled shortest path node $(i \rightarrow j)$ from step 2, set

$R_i^k = D_i^k - L^k$ and $K_j^k = L^k - D_j^k$. For any remaining origins i or destinations, j , set $R_i^k = K_j^k = 0$.

4. Set $c_{ij}^{k+1} = c_{ij}^k - R_i^k - K_j^k$ for $(i, j) \in E$.
5. Whenever the i -th shortest path node is in $S(v^k)$, $i \neq m + 1$, let $T = \min \left(b'_v - b_v, x_{i, X_{i,0}^k}, b_{d^k} - \sum_{(i, d^k) \in RS(d^k)} x_{id^k}^k \right)$ where $X_{i,0}^k$ is node j receiving $x_{i, X_{i,0}^k}^k$ units at step k . Set $r_i^{k+1} = r_i^k + 1$, and $X_{i, r_i^{k+1}}^{k+1} = X_{\ell, Q}^k$ if $x_{i, X_{i,0}^k}^k > T$. If $x_{i, X_{i,0}^k}^k = T$, set $X_{i,0}^{k+1} = X_{\ell, Q}^k$, where O and Q are the destination number for the source node i and ℓ , respectively, and $\ell = P_i^k$. For all other origins i , $X_{i,0}^{k+1} = X_{i,0}^k$. Modify supplies in $S(v^k)$ accordingly, and set $k \rightarrow k + 1$ and go to step 1 of the SSP algorithm.

In other words, step 5 identifies the amount by which the deficient node is short, by which the abundant node is in excess, and by which each node in the shortest path tree can ship. Then every node on the $S(v^k)$ path is examined to find how much is available in that node. Further, the minimum is taken over all the quantities and shipment is reversed along the path in order to reduce the deficiency in the root node.

Proof: Consider the following six cases:

1. If $(i \rightarrow j)$ is a node of $S(v^k)$.

$$\begin{aligned} R_i^k + K_j^k &= (D_i^k - L^k) + (L^k - D_i^k) = 0 \\ \Rightarrow R_i^k + K_j^k &\leq c_{ij} \end{aligned}$$

2. If there exists p and q such that $(p \rightarrow j)$ is the predecessor of $(i \rightarrow q)$ in $S(v^k)$. (adjacent nodes). Hence, $D_i^k = D_p^k + c_{ij}^k$. It follows that

$$\begin{aligned} R_i^k + K_j^k &= (D_i^k - L^k) + (L^k - D_p^k) = c_{ij}^k \\ \Rightarrow R_i^k + K_j^k &\leq c_{ij}^k \end{aligned}$$

3. If there exists p and q and $(p \rightarrow j)$ is not the predecessor of $(i \rightarrow q)$, in $S(v^k)$. (not adjacent nodes)

$$\begin{aligned} R_i^k + K_j^k &= (D_i^k - L^k) + (L^k - D_p^k) = D_i^k - D_p^k \\ \text{but, } D_i^k &< D_p^k \\ \Rightarrow R_i^k + K_j^k &\leq c_{ij}^k \end{aligned}$$

4. If node $(i \rightarrow q)$ in $S(v^k)$, but node $(p \rightarrow j)$ is not in $S(v^k)$ and permanently labelled.

$$\begin{aligned} R_i^k + K_j^k &= (D_i^k - L^k) + (L^k - D_p^k) = D_i^k - D_p^k \\ \text{but, } D_i^k &< D_p^k \\ \Rightarrow R_i^k + K_j^k &\leq c_{ij}^k \end{aligned}$$

5. If node $(i \rightarrow q)$ in $S(v^k)$, but node $(p \rightarrow j)$ is not in $S(v^k)$ and not yet permanently labelled.

$$\begin{aligned} R_i^k + K_j^k &= (D_i^k - L^k) + 0 = D_i^k - L^k \leq 0 \\ \Rightarrow R_i^k + K_j^k &\leq c_{ij}^k \end{aligned}$$

Otherwise, node $(p \rightarrow j)$ will be labelled before reaching the abundant node.

6. If node $(i \rightarrow q)$ is permanently labelled but $(p \rightarrow j)$ is not labelled.

$$\begin{aligned} R_i^k + K_j^k &= (D_i^k - L^k) + 0 \leq 0 \\ \Rightarrow R_i^k + K_j^k &\leq c_{ij}^k \end{aligned}$$

Otherwise, node $(p \rightarrow j)$ will be labelled before the affluent node is labelled.

The SSP algorithm maintains dual feasibility and complementary slackness and strives for primal feasibility. When it attains primal feasibility it stops.

Theorem 2. *If SSP does not stop because of the infeasibility test in step 2, it reaches optimality in at most $A - \min_j(b_j)$ shortest path step, where $A = \sum_{i=1}^m a_i = \sum_{j=1}^n b_j$.*

Proof: At each iteration the reduction in primal infeasibility is at least one unit. Therefore, the maximum number for shortest path steps is $A - \min_j(b_j)$.

Theorem 3. *SSP algorithm for the transportation problem has a $O((A + m)^3)$ computational bound where $A = \sum_i a_i = \sum_j b_j$.*

Proof: At iteration k the maximum number of node in the shortest path problem is $(k + m)$. The computational bound on the shortest path is $O(n^2)$, where n is the number of nodes in the shortest path tree. Therefore, the computational bound on the SSP for the transportation problems is

$$\sum_{r=(k+m)}^{(A-\min_j(b_j)+m)} r^2 = \sum_{r=1}^{(A-\min_j(b_j)+m)} r^2 - \sum_{r=1}^m r^2$$

Using the formula for the sum of square of integers and simplifying, the computational bound on the SSP algorithm is $\simeq O((A + m)^3)$.

Theorem 4. *The original transportation problem is infeasible if and only if the shortest path algorithm fails to label an abundant node of $SP(C^k, X^k, d^k)$ on some iteration k .*

Proof. Half the proof follows from theorem 2. Regarding the other half, we note that whenever a path from the root to some abundant node exists, the shortest path algorithm will eventually permanently label an abundant node. We proceed by assuming that there is no path from the root to an abundant node in $SP(C^k, X^k, d^k)$ and that the desired conclusion – infeasibility of the transportation problem – does not hold.

5.3.2 Implementation of the SSP Algorithm for the Transportation Problem

Going from the implementation of the SSP algorithm for the assignment and semi-assignment problem will not change much. The resulting shortest path problem is solved using a label-setting algorithm, Dijkstra implementation. The selection of deficient nodes is done similar to the case of the assignment and semi-assignment. The major changes are in the construction of the shortest path tree

and reversing the shipments along the path $S(v^k)$ (Step 5 of the SSP algorithm).

Deficient nodes with positive shipments can be a part of the $SP(C, X, d)$. The construction of these nodes and the linkage to the other nodes in the tree is the same as discussed in Chapter 3. We deal with these nodes similar to other nodes except we have to keep in mind that it is deficient by a specific amount.

When reversing the shipment along the path $S(v^k)$, some destination nodes may change their source of shipment, and some sources may reduce some of the shipment from a destination node. This case is explained in step 5 of the SSP algorithm and it depends on the amount T that will be taken from the abundant node to the predecessor of the abundant node and from the predecessor of the abundant node to the predecessor of the predecessor of the abundant node. This process is continued until we reach the root node. In order to fix the idea of reversing the shipment, we present an example. Consider a deficient node d^1 , where $b_{d^1} = 6$ and $\sum_{(i,d^1) \in RS(d^1)} x_{i,d^1}^1 = 4$. After solving the SSP. Steps 1 to 4 we found that the $S(v')$ is as shown in Figure 5.1. Step 5 of the SSP algorithm starts by identifying the amount T that will be shipped along the path, $T = \min(8 - 3, 8, 2, 3, 6 - 4) = 2$. Then the shipment is taken from the abundant node along the path $S(v')$ until we reach the root node. Therefore, by starting from the abundant node until we reach the root node the following are the results.

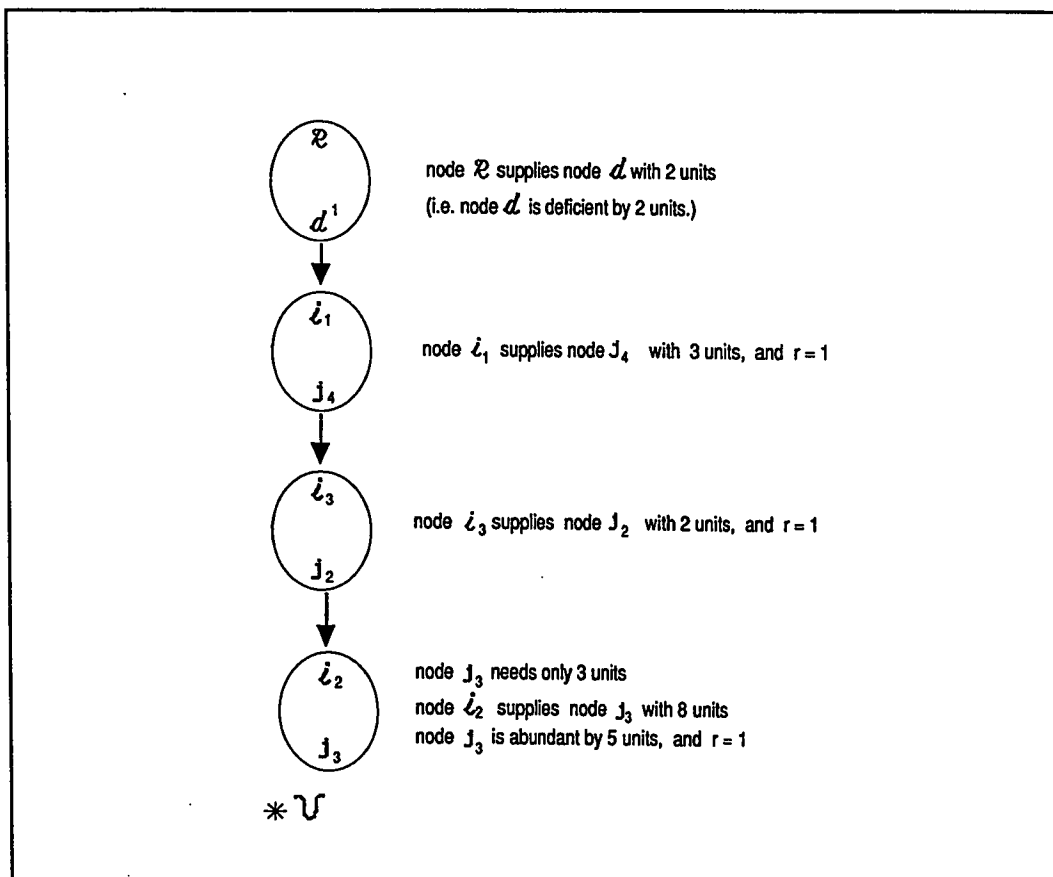


Figure 5.1 An Example of a Shortest Path ($S(v)$) to an Abundant Node from a Root Node

1. Because $x_{i_2 j_3}^1 = 8 > T$, $r_{i_2}^2 = 1 + 1 = 2$

Hence, $X_{i_2, 2}^2 = X_{i_3, 1}^1 = j_2$ and then $x_{i_2 j_3} = 8 - 2 = 6$ and $x_{i_2 j_2}^2 = 2$

2. Because $x_{i_3 j_2}^1 = 2 = T$, $r_{i_3}^2 = r_{i_3}^1 = 1$

Hence, $X_{i_3, 1}^2 = X_{i_1, 1}^1 = j_4$ and then $x_{i_3 j_2} = 2 - 2 = 0$ and $x_{i_3 j_4}^2 = 2$

3. Because $x_{i_1 j_4}^1 = 3 > T$, $r_{i_1}^2 = 1 + 1 = 2$

Hence, $X_{i_1, 2}^2 = X_{R, 1}^1 = d$ and hence $x_{i_1 j_4}^2 = 3 - 2 = 1$ and $x_{i_1 d}^2 = 2$

Then set $k = 2$ and go to step 1 of the SSP algorithm.

When an iteration k is completed it does not guarantee that the deficient node d^k is satisfied by the amount by which it is deficient. It might be worth while to develop a strategy for picking a deficient node in the SSP algorithm; however, some experimentation was done, but an efficient strategy was not formulated. Further work is needed to develop such a strategy.

5.4 Comparative Computational Tests

The SSP steps are coded using FORTRAN language and called SPTN. Computer environment and the comparison of SPTN with the available software are presented in this section.

5.4.1 Experimental Design

The computer environment used to test SPAN-I and SPSN is used here to test SPTN code. The problems used in the tests were randomly generated using NET-GEN [55]. The parameter specifications of the problems were picked to correspond with the problems used for comparisons in [8, 37].

All problems were solved on an AMDAHL 5850 using FORTRAN compiler. The computer jobs were executed during periods when the machine load was approximately the same, and all times are exclusive of input and output. The total time spent solving the problems was recorded by calling a CPU clock upon starting to solve the problem and again when the solution was obtained. The solution time recorded is an average of five runs for each problem.

Ten problems were generated. The first five problems are 100×100 of density varying from 13% to 29%, and the total supply of the 100 sources is 100,000 units. Problems 6-10 are 150×150 of density varying from 14% to 28%, and the total

supply of the 150 source is 150,000 units. The unit cost of these ten problems varies from 1 to 100. In order to see the sensitivity of the codes solution times to the value of the cost vector, the cost vector range was changed to be randomly generated from values between 1 and 10,000. Then the same ten problems are used again for comparison with the new cost range.

5.4.2 Computational Comparison

Two codes are used for computational comparison with SPTN. The codes are CAPNET and TRANS. The first code is an implementation of a specialized primal simplex [9] and the second code is an implementation of the Out-of-Kilter algorithm developed by Fulkerson [32]. The two codes are in-core and coded in FORTRAN.

The bench-mark problems obtained from [55] are solved using the three codes. The CPU times for solving these problems are shown in Tables 5.1 and 5.2. Table 5.1 shows the problem size, the problem density (in terms of the number of arcs), total supply, and solution times (in seconds). Table 5.2 contains the same information contained in Table 5.1 for the same set of problems with a different cost range. Graphical representations of the solution times for SPTN and CAPNET are given in Figures 5.2 and 5.3

Based on the sum of the solution times for the problems in Table 5.1 with cost

range 1–100, SPTN is roughly 1.3 times faster than CAPNET, and the solution times for TRANS is not comparable with SPTN solution times. For the cost range from 1 to 10,000 (in Table 5.2), based on the sum of the solution times, it was found that CAPNET is roughly 1.23 times faster than SPTN. Because of memory limitations TRANS could not solve any problem when the cost range is between 1 and 10,000. Also, it is noticed that the change in the cost range did not alter CAPNET CPU solution times.

The study indicates that the SPTN code is more efficient than CAPNET for problem with low cost range. However, the results in Table 5.2 shows that CAPNET is more efficient than SPTN when the cost range is high. The reason that CAPNET dominates SPTN in this case, because of the length of the sort list array (K) needed for solving the shortest path problem in SPTN which depends on the maximum cost value.

TABLE 5.1 Total Solution Times in Seconds for Transportation Problems with a cost range of 1-100.

No. of Nodes <i>m</i> × <i>n</i>	No. of arcs	Total Supply	Times (seconds)		
			CAPNET	TRANS	SPTN
100 × 100	1300	100,000	0.830	42.68	0.7196
100 × 100	1500	100,000	0.868	40.89	0.7829
100 × 100	2000	100,000	1.007	31.87	0.8313
100 × 100	2200	100,000	1.100	28.27	0.8045
100 × 100	2900	100,000	1.285	21.05	0.8466
150 × 150	3150	150,000	1.597	126.46	1.086
150 × 150	4500	150,000	2.077	109.83	1.515
150 × 150	5155	150,000	2.223	87.53	1.808
150 × 150	6075	150,000	2.513	69.24	1.906
150 × 150	6300	150,000	2.532	69.95	2.039
Total Solution Times			16.032	620.77	12.329
Ratio			1.30	50.3	1

TABLE 5.2 Total Solution Times in Seconds for Transportation Problems with a cost range of 1–10,000.

No. of Nodes $m \times n$	No. of arcs	Total Supply	Times (seconds)		
			CAPNET	TRANS	SPTN
100 × 100	1300	100,000	0.829	CNT	1.043
100 × 100	1500	100,000	0.867	CNT	1.251
100 × 100	2000	100,000	1.033	CNT	1.608
100 × 100	2200	100,000	1.047	CNT	1.472
100 × 100	2900	100,000	1.283	CNT	1.848
150 × 150	3150	150,000	1.592	CNT	1.560
150 × 150	4500	150,000	2.086	CNT	2.214
150 × 150	5155	150,000	2.251	CNT	2.683
150 × 150	6075	150,000	2.502	CNT	2.984
150 × 150	6300	150,000	2.526	CNT	3.161
Total Solution Times			16.016	—	19.824
Ratio			1	—	1.23

CNT = could not run as a result of memory limitations

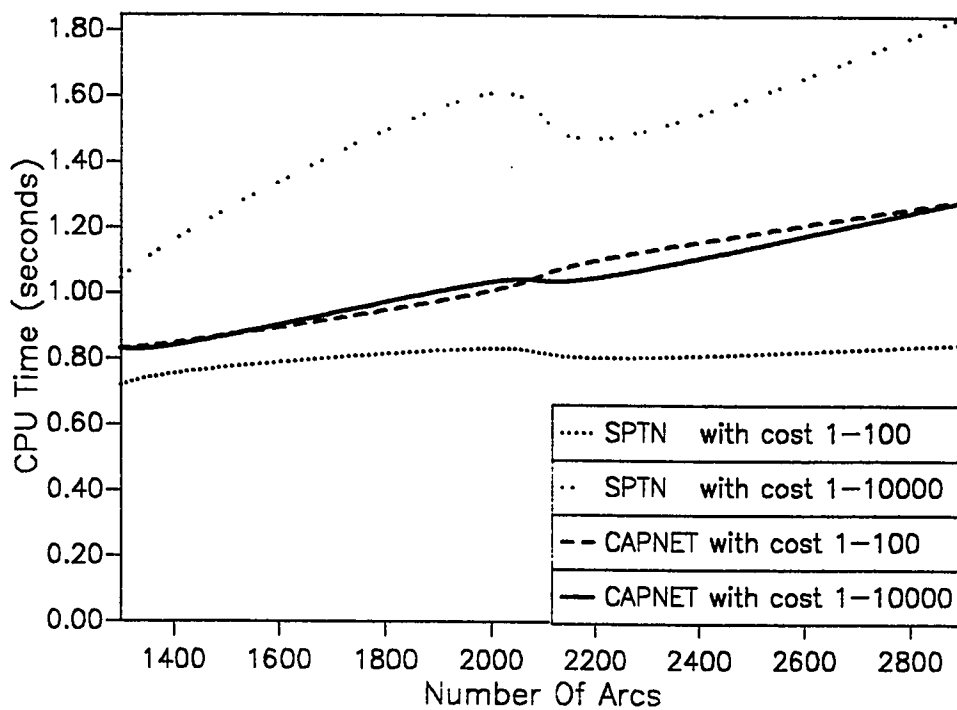


Figure 5.2 100×100 Transportation Problems with Different Cost Range and Total Supply = 100,000

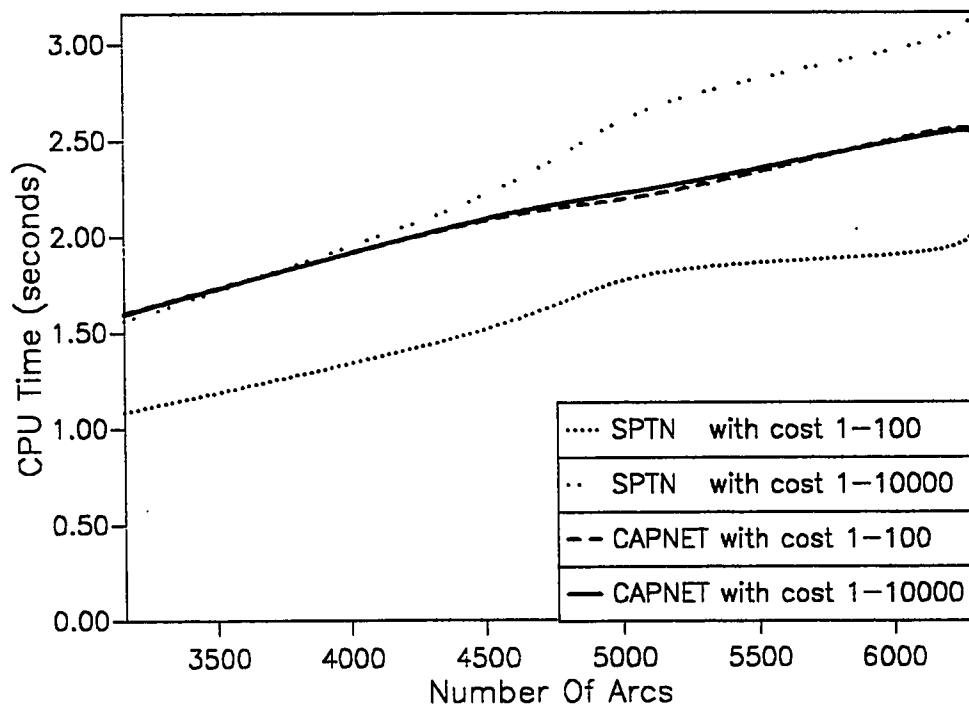


Figure 5.3 150 × 150 Transportation Problems with Different Cost Range and Total Supply = 150,000

5.4.3 Memory Requirements of the Codes

Table 5.3 shows the developer names, code name, type of the algorithm used in each code and the number of arrays needed for each code for solving the transportation problem. The storage requirements of TRANS code were not available. Based on the tested problems, CAPNET has a relatively less memory requirements than SPTN.

TABLE 5.3 Code Specifications for Solving the Transportation Problem

Developer	Name	Type	Number of arrays
1. Duffuaa and Ghassab	SPTN	SSP algorithm	$10M + 4N + 4A + 2L$
2. SAS	TRANS	Out-of-Kilter Method	Not available
3. Barr <i>et al.</i>	CAPNET	Specialized Primal Simplex algorithm	$6M + 6N + 3A$

where,

M = number of origins

N = number of destinations

A = number of arcs

L = maximum arc length

5.5 Conclusion

The SSP algorithm discussed in this chapter is a generalized SSP algorithm for solving the transportation problem. After coding this SSP algorithm and comparing it with the available software results indicated that SPTN is 1.3 times faster than CAPNET for transportation problem with cost range between 1–100; however, CAPNET is 1.23 times faster than SPTN for problems with cost range between 1–10,000. Also, based on memory requirement, CAPNET has a relatively less memory requirements than SPTN.

CHAPTER 6

CONCLUSIONS

6.1 Summary of Results

Application of minimal cost network flow problems are widely spread, [35, 36, 38] and new solution algorithms and implementations have stimulated further applications. A number of studies have concluded that implementations based on the primal simplex algorithm are the most efficient for solving such problems.

In this thesis we have introduced a successive shortest path algorithm – SSP – and an implementation of this algorithm – SPAN-I. We have verified through computational testing that Dijkstra algorithm is the best shortest path algorithm which fits the SSP framework. SPAN-I was also found to be more efficient than the implementations of Hungarian and specialized primal simplex algorithms for sparse problems.

An SSP algorithm for semi-assignment problem is developed and implemented in the code SPSN. Computational comparisons have been performed with a specialized primal simplex code (CAPNET), an implementation of Out-of-Kilter algorithm (TRANS) and solving semi-assignment as an assignment problem (SPSAN).

It has been verified that SPSN is the most efficient algorithm for solving semi-assignment problems. It is 1.3 times faster than CAPNET and 1.8 times faster than SPSAN.

Finally, the SSP approach is generalized for solving uncapacitated transportation problems. A code implementing the SSP for transportation problem is developed (SPTN). It has been verified through computational testing that SPTN is more efficient for solving uncapacitated transportation problems than CAPNET, for problems with cost range between 1 and 100. For such problems it is 1.3 times faster than CAPNET. For problems with high cost range between 1 and 10,000, the specialized primal simplex code (CAPNET) has dominated the code (SPTN) developed in this thesis. For such problems CAPNET is 1.23 times faster than SPTN.

6.2 Conclusions

Concluding the work of this thesis, the following are the main findings:

1. The best shortest path algorithm which fits the SSP framework is a label-setting algorithm using Dijkstra implementation.
2. One of the best algorithms for solving sparse assignment problems is SSP algorithm. However, for 100% dense problems the Hungarian method performs better than SSP algorithm.
3. The modified SSP algorithm for solving semi-assignment problems is one of the most efficient algorithms, in comparison with the method used and solution times reported in the literature.
4. For transportation problems with low cost range of 1-100, the generalized SSP algorithm for solving uncapacitated transportation problems is verified to be one of the best algorithms. However, if the cost range is between 1-1000, CAPNET outperformed the SSP algorithm.
5. In general SSP algorithms tend to be more efficient when the range of the cost vector is small. This is expected because in this case the required list of address calculation sort has smaller dimensions and requires less data manipulations.

6.3 Future Research

The following are some areas for future research:

1. Extension and implementation of the SSP approach for capacitated transportation problem might be worthwhile.
2. New theoretical results for the generalized SSP algorithm have been developed. However, it would be of interest if a more general framework could be set up which would relate the thesis work to the family of simplex algorithms.
3. Strategies for selecting deficient nodes for constructing the shortest problem may influence algorithm efficiency and need to be investigated.
4. Large cost variations increase the CPU time for SSP algorithm because the address calculation sort dimension in solving the SP problem depends on the maximum cost. Therefore, scaling of the cost range that will reduce the CUP time needs to be investigated.

REFERENCES

1. Ahuja, R., T. Magnanti, and Orlin, J., "Network Flows", *In Handbooks in Operations Research and Management Science*, 1989.
2. Ahuja, R., T. Magnanti, and Orlin, J., "Some Recent Advances in Network Flows", *SIAM Review*, 33(2), 1991, pp. 175-219.
3. Ahuja, R., T. Magnanti, and Orlin, J., *Network Flows*, A Simon & Schuster Company, 1993.
4. Balinski, M., "Signature Methods for the Assignment Problem", *Operation Research*, 33(3), pp. 527-536, 1985.
5. Balinski, M., "A Competitive (Dual) Simplex Method for the Assignment Problem", *Mathematical Programming*, 34, pp. 125-141, 1986.
6. Balinski, M., and Gomory, R., "A Primal Method for the Assignment and Transportation Problem", *Management Science*, 10(3), pp. 578-593, 1964.
7. Barr, R., Glover, F. and Klingman, D., "An Improved Version of the Out-of-Kilter Method and a Comparative of Computer Codes", *C.S. Report 102*, Center for Cybernetic Studies, University of Texas at Austin.
8. Barr, R., Glover, F. and Klingman, D., "A New Alternating Basis Algorithm for Semi-Assignment Networks", *Research CCS 264*, Center for Cybernetic Studies, The University of Texas at Austin (1977).
9. Barr, R., Glover, F., and Klingsman, D., "A Code for the Specialized Primal Simplex Method", *University of Texas at Austin*.
10. Barr, R., Glover, F. and Klingman, D., "The Alternating Basis Algorithm for Assignment Problems", *Mathematical Programming 13*, pp. 1-13, 1977.
11. Barr, R., Glover, F. and Klingman, "The Generalized Alternating Path Algorithm for Transportation Problems" *European Journal of Operations Research* 2(2), pp. 134-144, 1978.
12. Barr, R., Glover, F. and Klingman, "A Network Augmenting Path Basis Algorithm for Transshipment Problems", *Research Report CCS 272* Aug., 1978.
13. Bazaraa, M., Jarvis, J. and Sherali, H., "Linear Programming and Network Flows", USA, 1990.

14. Bellman, R., "On a Routing Problem", *Quarterly Applied Mathematics*, 16(1), pp. 87-90, Apr., 1958.
15. Bertsekas, D., "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem", *Annals of Operations Research*, 14, pp. 105-123, 1988.
16. Carpaneto, G., and Paolo, T., "Algorithm 548-Solution of the Assignment Problem", *ACM Transactions on Mathematical Software*, 6(1), pp. 104-111, 1980.
17. Charnes, A. and Cooper, W., "The Stepping Stone Method of Explaining Linear Programming Calculations in Transportation Problems", *Management Science*, 1(1), pp. 49-69, 1954.
18. Current, J., Revelle, C. and Cohon, J., "The Median Shortest Path Problem: A Multiobjective Approach to Analyze Cost vs. Accessibility in the Design of Transportation Networks", *Transportation Science*, 21(3), pp. 188-197, 1987.
19. Cunningham, W.H., "A Network Simplex Method", *Mathematical Programming*, 11(2), pp. 105-116, 1976.
20. Dantzig, D., *Linear Programming on Extensions*, Princeton University Press, Princeton, N.J., 1963.
21. Dantzig, G., "Application of the Simplex Method to a Transportation Problem", *Activity of Production and Allocation*, John Wiley & Sons, New York, pp. 359-373, 1951.
22. Dantzig, G., "On the Shortest Route Through a Network", *Management Science*, 6(2), pp. 187-190, 1960.
23. Dial, R. and Voorhees, A., "Algorithm 360-Shortest Path Forest with Topological Ordering", *Communications of the ACM*, 12(11), pp. 632-633, Nov. 1969.
24. Dial, R., Glover, F., Karney, D. and Klingman, D., "A Computational Analysis of Alternative Algorithms and Labelling Techniques for Finding Shortest Path Trees", *Networks*, 9, pp. 215-248, 1979.
25. Dijkstra, E., "A Note on Two Problems in Connection with Graphs", *Numerische Mathematik I*, 269-271, 1959.
26. Dinic, E. and Kronrod, M., "An Algorithm for the Solution of the Assignment Problem", *Soviet Math. Doklady*, 10(6), 1969.

27. Divoky, J. and M. Hung, "Performance of Shortest Path Algorithms in Network Flow Problems", *Management Science*, 36(6), pp. 661-673, 1990.
28. Dreyfus, S., "An Appraisal of Some Shortest-Path Algorithms", *Operations Research*, 17(3), pp. 395-412, 1969
29. Edan, Y., Flash, T., Peiper, U., Shmulevich, I. and Sarig, Y., "Near-Minimum-Time Task Planning for Fruit Picking Robots", *IEEE Transactions on Robotics and Automation*, 17(1), pp. 48-56, 1991.
30. Enguist, M., "A Successive Shortest Path Algorithm for the Assignment Problem", *INFOR*, 20(4), pp. 370-384, 1982.
31. Floyd, R.W., "Algorithm 97: Shortest Path", *Communication of ACM*, 5(6), p. 345, 1962.
32. Fulkerson, D.R., "An Out-of-Kilter Method for Minimal Cost Flow Problems", *Journal of the Society for Industrial and Applied Mathematics*, 9(1), 1961.
33. Gabow, H., and R. Tarjan, "Faster Scaling Algorithms for Network Problems", *SIAM Journal on Computing*, 18(5), pp. 1013-1036, 1989.
34. Gilsinn, J. and Witzgall, C., "A Performance Comparison of Labelling Algorithms for Calculating Shortest Path Trees", *NBS Technical Note 772*, U.S. Dept. of Commerce, 1973.
35. Glicksman, S., Johnson, L., and Eselson, L., "Coding the Transportation Problem", *Naval Research Logistics Quarterly*, 7, pp. 169-183, 1960.
36. Glover, F., Karney, D., and Klingman, D., "Implementation and Computational Comparisons of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems", *Research Report CCS 136*, The University of Texas at Austin, July 1973.
37. Glover, F. Karney, D., Klingman, D. and Napier, A., "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems", *Management Science*, 20(5), pp. 793-813, 1974.
38. Glover, F., and Klingman, D., "Network Application in Industry and Government", *Center for Cybernetic Studies*, The University of Texas at Austin, 1975.
39. Glover, F., Klingman, D. and Phillips, N., and Schneider, R., "New Polynomial Shortest Path Algorithms and Their Computational Attributes", *Management Science*, 31(9), pp. 1106-1128, 1985.

40. Glover, F., Klingman, D. and Phillips, N., "A New Polynomially Bounded Shortest Path Algorithm", *Operations Research*, 33(1), pp. 65-73, 1985.
41. Glover, F., Hultz, J., and Klingman, D., "Improved Computer-based Planning Techniques, Part I", *Interfaces*, 8(4), pp. 16-24, 1978.
42. Glover, F., Hultz, J., and Klingman, D., "Improved Computer-Based Planning Techniques, Part II", *Interfaces*, 9(4), pp. 12-20, 1979.
43. Glover, F., Glover, R., and Klingman, D., "Threshold Assignment Algorithm", *Mathematical Programming Study*, 26, pp. 12-37, 1986.
44. Goldfarb, D., "Efficient Dual Simplex Algorithms for the Assignment Problem", *Mathematical Programming*, 33, pp. 187-203, 1985.
45. Gribov, A., "A Recursive Solution for the Transportation Problem of Linear Programming", *Vestnik of Leningrad University*, 33, pp. 37-52, (in Russian), 1978.
46. Hitchcock, F.L., "Distribution of a Product from Several Sources to Numerous Localities", *Journal of Mathematical Physics*, 20, pp. 224-230, 1941.
47. Hung, M. and J. Divoky, "A Computational Study of Efficient Shortest Path Algorithms", *Computer and Operations Research*, 15(6), pp. 567-576, 1992.
48. Hung, M., and Rom, W., "Solving the Assignment Problem by Relaxation", *Operations Research*, 28(4), pp. 969-982, 1980.
49. Jensen, P. and J. Barnes, *Network Flow Programming*, John Wiley & Sons, 1987.
50. Johnson, D., "A Note on Dijkstra's Shortest Path Algorithm", *Journal of the Association for Computing Machinery*, 20(3), pp. 358-388, 1973.
51. Johnson, E., "Networks and Basic Solutions", *Operations Research*, 14, pp. 819-623, 1966.
52. Jonker, R. and Volgenant, T., "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems", *Computing*, 38, pp. 325-340, 1987.
53. Kennington, J. and Helgason, R., *Algorithms for Network Programming*, John Wiley & Sons, 1980.
54. Kennington, J. and Wang, Z., "A Shortest Augmenting Path Algorithm for the Semi-Assignment Problem", *Operations Research*, 40(1), pp. 178-187, 1992.

55. Klingman, D., Napier, A., and Stutz, J., "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems", *Management Science*, 20(5), pp. 814–821, 1974.
56. Klingman, D., Mote, J., and Whitman, D., "Improving Flow Management and Control via Improving Shortest Path Analysis", *Research Report CCS 322*, The University of Texas at Austin, 1978.
57. Klingman, D. and Schneider, R., "Microcomputer-Based Algorithms for Large Scale Shortest Path Problems", *Discrete Applied Mathematics*, 13, pp. 183–206, 1986.
58. Kuhn, H., "The Hungarian Method for the Assignment Problem", *Naval Research Logistic Quarterly*, 2, pp. 83–97, 1955.
59. McGinnis, L.F., "Implementation and Testing of a Primal–Dual Algorithm for the Assignment Problem", *Operations Research*, 31(2), pp. 277–291, 1983.
60. Orlin, J., and R. Ahuja, "New Scaling Algorithm for the Assignment and Minimum Mean Cycle Problems", *Mathematical Programming*, 54, pp. 41–56, 1992.
61. Pape, U., "Implementation and Efficiency of Moore–Algorithm for the Shortest Route Problem", *Mathematical Programming*, 7, pp. 212–222, 1974.
62. Pape, U., "Algorithm 562–Shortest Path Lengths", *ACM Transactions on Mathematical Software*, 6(3), pp. 450–455, 1980.
63. *SAS Operations Research Manual*, The ASSIGN Procedure, Chapter 4, pp. 55–65, 1992
64. *SAS Operations Research Manual*, The TRANS Procedure, Chapter 10, pp. 443–456, 1992.
65. Shier, D., "On Algorithms for Finding the K Shortest Path in a Network", *Networks*, 9, pp. 195–214, 1979.

APPENDICES

Appendix A

A.1 Implementation Techniques for the Label-Setting Method

In this appendix, several implementations of the general label-setting method are discussed. From an algorithmic viewpoint, the primary differences between these implementations are the way in which the minimum in step 3 of the algorithm description is found and the handling of original problem data.

A naive implementation of the general label-setting method would be to find the set S of step 2 by examining all arcs in A and then calculating and discarding node potentials to find the minimum of step 3. This involves examining all arcs during every execution of step 2, as well as performing many unnecessary node potential calculations in step 3. The implementations described in this subsection make use of temporarily retained node potentials in such a way that each arc in A is examined *at most once*, thereby avoiding extensive recalculation.

As a basis for understanding these implementations, it is useful to observe that steps 2 and 3 of the label-setting method simply find an arc from a tree node to a non-tree node which yields the minimum distance extension. Figure A.1 illustrates one way of viewing these steps at some iteration where the tree $T(N_T, A_T)$ consists of the *solid line* arcs and their associated nodes. The *dashed line* arcs and their ending nodes N_E indicate possible tree extensions. (Note that $N - N_T$ may not be equal to N_E).

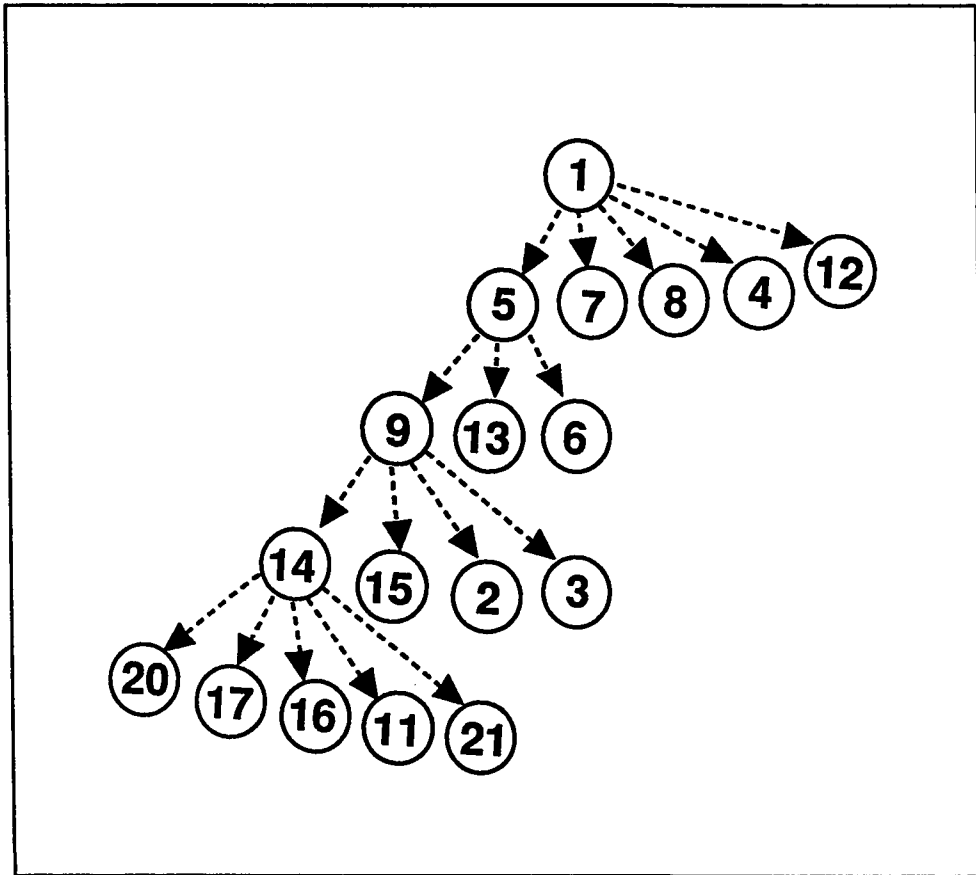


Figure A.1 Label-Setting Iteration

By reference to this diagram, it may be seen that steps 2 and 3 can be performed by keeping a temporary node potential and predecessor for each one v in N_E such that $d(v) = \min_{u \in N_T} (d(u) + \ell(u, v))$ and the predecessor of v is set to a node u which yields the minimum node potential for v . Thus, if $p(v) = u$ then $-d(u) + d(v) = \ell(u, v)$. Step 3 then adds a node v in N_E with the smallest temporary node potential to N_T and correspondingly adds its arc $(p(v), v)$ to A_T . After performing this step, node v 's potential will never change (*i.e.*, it is assigned a *permanent node potential* at this time) and arc $(p(v), v)$ is permanently assigned to the tree. The name label-setting stems from this property of the algorithm.

In the following subsections, we discuss alternative implementations for carrying out steps 2 and 3 in this manner. These implementations differ in the way they handle the following fundamental operations: (1) the computation and updating of temporary node potentials, (2) the assignment of one or more temporary node potentials to a node in N_E , (3) the representation of the original network on the external file, and (4) the buffering of arc data into central memory.

A.1.1 Dijkstra Address Calculation Sort

The first implementation to be discussed is the one originally developed by Dial [23]. The Dial implementation then identifies the minimum temporary node potential using the following observation. Each temporary node potential equals a permanent node potential plus the length of some arc. Consequently, temporary

node potential values may be *uniquely* represented modulo $(\ell_{\max} + 1)$ where $\ell_{\max} = \max_{a \in A} \ell(a)$. That is if $d(p) \neq d(q)$ where $d(p)$ and $d(q)$ are *temporary node potentials*, then $d(p)$ modulo $(\ell_{\max} + 1) \neq d(q)$ modulo $(\ell_{\max} + 1)$.

To see this, suppose that node v has the minimum temporary node potential at the current iteration. Then $d(u) \leq d(v)$ for $u \in N_T$ and thus for $t \in N_E$ $d(v) \leq d(t) \leq d(v) + \ell_{\max}$. In other words, at each iteration all temporary node potentials are bracketed on the lower side by $d(v)$ and on the upper side by $d(v) + \ell_{\max}$. Thus it is possible from one iteration to the next to uniquely represent all temporary node potentials modulo $(\ell_{\max} + 1)$.

To find the minimum by this procedure, it is convenient to use a computer array k of size $\ell_{\max} + 1$ where

$$k(i) = \begin{cases} 0 & \text{if } i \neq d(v) \text{ modulo } (\ell_{\max} + 1), \text{ for any } v \in N_E \\ p_i & \text{if } i = d(q) \text{ modulo } (\ell_{\max} + 1), \text{ for some } q \in N_E \end{cases}$$

where p_i is a pointer which points to all nodes in N_E that have a modulo temporary node potential value of i . The nodes in N_E that have the same modulo temporary node potential value (and thus, on any given iteration, the same temporary node potential value) are identified by chaining the nodes by a two-way linked list. Thus, every node with the same temporary potential value is linked to an antecedent and a successor node (which may be dummies at the "ends" of the list). When a node's temporary potential changes, the node is disconnected from the chain simply by re-linking this antecedent and successor to each other. This array achieves an

“automatic sort” of the nodes in N_E relative to their temporary node potentials. Figure A.2 illustrates the sort structure induced by the k array and the two-way linked lists, representing node names by the symbol n_i .

The current minimum temporary node potential is found by sequentially examining the elements of k in a wrap-around fashion. Each time a nonzero element of k is encountered, the current minimum node potential is that of the nodes associated with this element, and examination of k resumes at the next nonzero element of k on the next iteration.

To describe the implementation of this algorithm, it is convenient to define the following terms:

1. The *imputed node potential value of node q* , relative to the forward star of v , denoted by $d_v(q)$, is $d(v) + \ell(v, q)$.
2. An *improving imputed node potential $d_v(q)$* is one such that $d_v(q) < d(q)$; *i.e.*, $d_v(q)$ is smaller than the current minimum temporary node potential of node q .
3. Node q is an *improving node* relative to $FS(v)$ if it has an improving imputed node potential.
4. A node v is *scanned* by examining $FS(v)$ and updating $d(q)$ and $p(q)$ for each improving node $q \in FS(v)$; *i.e.*, $d(q) := d_v(q)$ and $p(q) = v$.

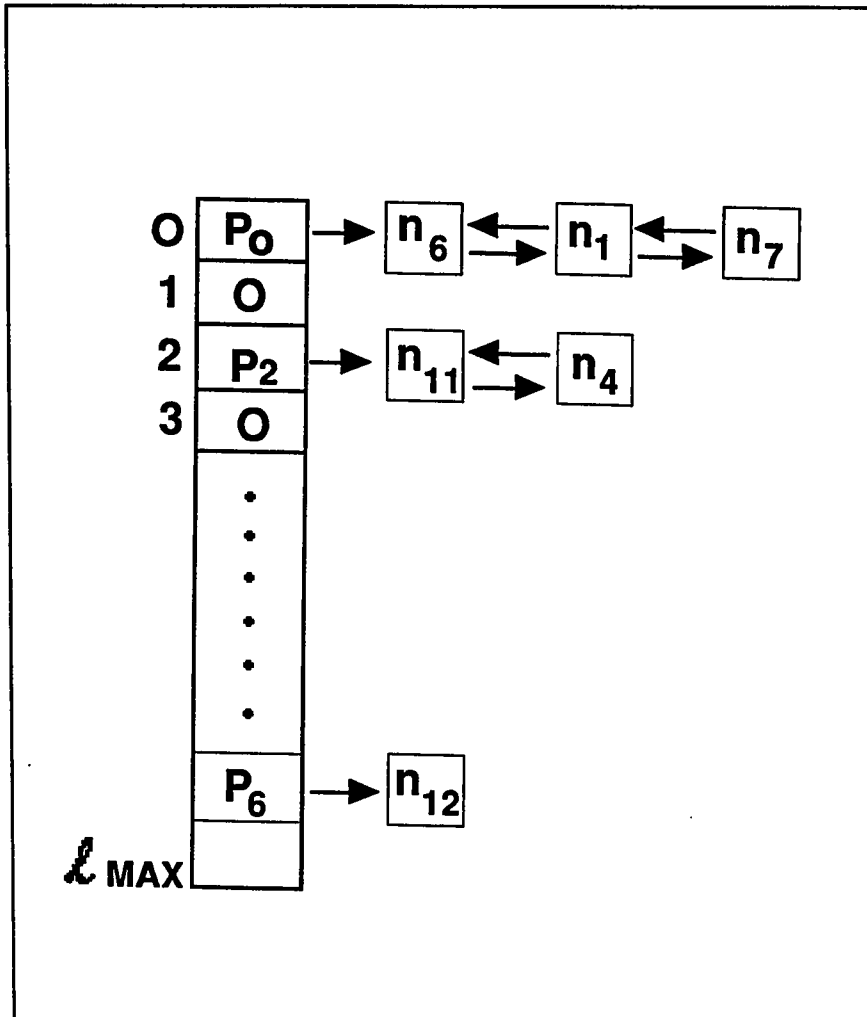


Figure A.2 Address Calculation Sort

To implement this approach, the algorithm initializes $p(v) = 0$, $v \in N$; $d(r) = 0$ and $d(v) = \infty$, $v \in N - \{r\}$; and $k(i) = 0$, $0 \leq i \leq \ell_{max}$. The root node r is then scanned and the improving nodes of $FS(r)$ are “added to” the appropriate elements of k . The first pass of the k list starts at $k(0)$, examining the elements of k in sequence until the first nonzero element is encountered. Each node v associated with this nonzero element is then removed from the two-way chained list and sequentially scanned. Any improving node q located during the scan of v is removed from “its current position” in k and moved to its new position $d_v(q)$ modulo $(\ell_{max} + L)$. (If $d(q) = \infty$ then node v has never been added to k and thus no steps are required to remove it.)

At each subsequent iteration, the examination of array k resumes where it left off (and wraps around if necessary) to find the first nonzero entry. This entry identifies a node with the new minimum temporary node potential. All chained nodes with this temporary node potential are then removed from k and scanned in the manner previously indicated. The algorithm stops when a complete pass of k is made without finding a nonzero entry.

This approach is called an *address calculation sort* because the insertion and deletion of an item from the list simply involves calculating an address in a convenient and straight-forward manner. Its application to shortest path implementations, as proposed and coded by Dial, is known in the literature as CACM Algorithm 360 (see [23]). This algorithm was found by authors of several studies

to be the most efficient shortest path method for problems with nonnegative arc lengths.

Two attractive features of this algorithm are its simplicity and the structuring which assures that each arc is examined at most once. The major disadvantages of this algorithm are the computer memory required to store k and the random access required of arc data.

A.1.2 Dantzig Address Calculation Sort

The study of [24] shows that a major time consuming task of Dijkstra implementation involves inserting and deleting nodes in the two-way linked array when their node potentials are reduced. One way to reduce the effort of inserting and removing nodes on the two-way linked list of the address calculation sort is to postpone adding nodes to the list. This can be done by observing that it is unnecessary to scan the entire forward star of the node v when it is assigned a permanent node potential. In particular, only the endpoint of a minimum length arc in such a forward star needs to be considered for addition to k . This follows from the fact that all temporary node potentials determined from node v will be greater than or equal to the node potential determined for the endpoint of a minimum length arc of $FS(v)$. We now describe an approach designed to exploit this observation.

In order to limit the nodes considered for addition to k by selecting a minimum length arc form $FS(v)$, it is convenient to store the network $G(N, A)$ in a *sorted forward star form*. Dantzig [22] was the first to suggest this type of scheme, and it is referred to as the Dantzig address calculation sort.

The steps of the algorithm basically operate in the manner previously described for Dijkstra except that: (A) the two-way linked list is replaced by a one-way linked list. (B) The forward star of each node u in N_T is scanned *until an improving node v is found*, whereupon v is placed on the linked list with its predecessor, u , and $p(v)$ is set to u and $d(v)$ is set to $d(p(v)) + \ell(p(v), v)$. (Node $p(v)$ is not scanned again until the ordered pair $(p(v), v)$ is removed from the linked list.)

It should be noted in this implementation, however, that the next nonzero element of k may not point to the next minimum. Thus, when a node v is removed from the linked list, it is discarded if its paired predecessor differs from its current predecessor in array p , since this implies that v has already been assigned a permanent node potential. In any event, the predecessor paired with v is scanned for its improving node. If an improving node is found, it is added to the linked list in the manner already described.

In the case that v 's paired predecessor is equal to its current predecessor $p(v)$, then v 's temporary node potential is a minimum and v is assigned a permanent potential and added to N_T . Further, node v is scanned as described in step B.

The advantages of this implementation are: (1) the algorithm can be terminated when all nodes are permanently labeled; (2) a node is never moved on the linked list when its node potential is improved; and (3) the postponement of adding temporary node potentials to k keeps less information on k and potentially avoids adding dominated values to k .

The materials in this Appendix are summarized from [24]. For more details on Dijkstra and Dantzig address calculation sort lists, see [22, 24].

Appendix B

C
 C THIS PROGRAM SOLVE THE ASSIGNMENT PROBLEM AS A SUCCESSIVE
 C SHORTEST PATH ALGORITHM USING DIJKSTRA ALGORITHM TO SOLVE
 C THE SHORTEST PATH PROBLEM.
 C

```

PROGRAM SPAN_I
PARAMETER (N=100, LA=10000, IH=100)
INTEGER PN(N+1), A(N+1), AJ(N), R(N), K(N), TN(LA)
INTEGER RN, P(N+1), V, L, PT, CT, Q, IV, DF, DK, LABEL(N), LB
INTEGER AL(LA), TOTC, D(N+1), DEF1(N), COST, CP
INTEGER NEXT(N+1), LAST(N+1)
INTEGER HEAD(0: IH), TAIL(0: IH)
LOGICAL SWAP, HOLD
CHARACTER FROM*10, TO*10, TITLE*35
EXTERNAL CTIME
DATA (K(I), I=1, N)/N*0/
DATA (AJ(I), I=1, N)/N*0/
READ(1, *) FROM, TO
TITLE = FROM
L1 = INDEX(TITLE, ' ')
L2 = INDEX(TO, ' ')
TITLE(L1+1: L1+L2) = TO
L3 = L1 + L2 + 1
TITLE(L3:) = 'ASSIGNMENTS'
II = 0
ML = -999
KK = 99999
PN(1) = 1
IM = 0

```

C
 C IN THIS SECTION THE DATA HAS TO BE READ AS :
 C FROM NODE, TO NODE, AND COST
 C

```

10 READ(1, *, END=20) I, J, COST
IM = IM + 1
TN(IM) = J
AL(IM) = COST
IF(II.NE.I) R(I) = 999999
IF(II.NE.I) PN(I+1) = PN(I)
PN(I+1) = PN(I+1) + 1
IF(COST.LT.R(I)) R(I) = COST
IF(COST.LT.KK) KK = COST
IF(COST.GT.ML) ML = COST
II = I
GOTO 10

```

C
 C BRINGING THE PROBLEM INTO THE STANDARD FORM
 C

```

20 T0 = CTIME()
DO 40 I=1, N
HOLD = .TRUE.
DO 30 JK=PN(I), PN(I+1)-1, 1
J = TN(JK)
COST = AL(JK) - R(I)
IF(COST.EQ.0.AND.HOLD) THEN

```

```
      A(I) = J
      AJ(J) = AJ(J) + 1
      HOLD = .FALSE.
    ENDIF
30  CONTINUE
40  CONTINUE
    RN = N + 1
    CP = ML
    ML = CP + 1
    DK = 1
C
C CHOOSEING THE DEFICIENT NODES
C
    DO 50 I=1,N
      IF(AJ(I).EQ.0) THEN
        DEFI(DK) = I
        DK = DK + 1
      ENDIF
50  CONTINUE
    DK = DK - 1
    DF = 1
    SWAP = .TRUE.
C
C SOLVING THE ASSIGNMENT PROBLEM
C
60  IF(DF.GT.DK.OR.(.NOT.SWAP)) GOTO 160
C
C INITIALLIZATION
C
    A(RN) = DEFI(DF)
    DO 70 I=1,ML-1
      HEAD(I) = 0
      TAIL(I) = 0
70  CONTINUE
    DO 80 I=1,N
      D(I) = 99999
80  CONTINUE
    PT = 0
    D(RN) = 0
    NEXT(RN) = 0
    LAST(RN) = 0
    P(RN) = 0
    HEAD(0) = RN
    I = HEAD(0)
    TAIL(0) = RN
    SWAP = .FALSE.
    LB = 0
    L = 0
    IV = 0
90  IF(I.EQ.0) GOTO 160
    IF(I.NE.RN) LABEL(LB) = I
    IF(I.NE.RN.AND.AJ(A(I)).GT.1) THEN
      SWAP = .TRUE.
      IV = I
      L = D(I)
```



```
GOTO 130
```

```
ENDIF
```

```
C
```

```
C SHORTEST PATH ALGORITHM (DIJKSTRA ADDRESS CALCULATION SORT)
```

```
C (LABEL SETTING ALGORITHM)
```

```
C
```

```
DO 110 NP=1,N
```

```
IF(I.EQ.NP) GOTO 110
```

```
DO 100 K1=PN(NP),PN(NP+1)-1,1
```

```
IF(TN(K1).NE.A(I)) GOTO 100
```

```
COST = AL(K1) - R(NP) - K(A(I))
```

```
V = D(I) + COST
```

```
J = NP
```

```
IF (V.LT.D(J)) THEN
```

```
IF (D(J).NE.99999) THEN
```

```
Q = MOD(D(J),ML)
```

```
IF (HEAD(Q).EQ.J) THEN
```

```
HEAD(Q) = NEXT(J)
```

```
ELSE
```

```
IF(TAIL(Q).EQ.J) THEN
```

```
TAIL(Q) = LAST(J)
```

```
NEXT(LAST(J)) = 0
```

```
ELSE
```

```
LAST(NEXT(J)) = LAST(J)
```

```
NEXT(LAST(J)) = NEXT(J)
```

```
ENDIF
```

```
ENDIF
```

```
ENDIF
```

```
Q = MOD(V,ML)
```

```
IF (HEAD(Q).EQ.0) THEN
```

```
HEAD(Q) = J
```

```
LAST(J) = 0
```

```
ELSE
```

```
LAST(J) = TAIL(Q)
```

```
NEXT(TAIL(Q)) = J
```

```
ENDIF
```

```
P(J) = I
```

```
D(J) = V
```

```
TAIL(Q) = J
```

```
NEXT(J) = 0
```

```
ENDIF
```

```
100 CONTINUE
```

```
110 CONTINUE
```

```
IF (NEXT(I).NE.0) THEN
```

```
I = NEXT(I)
```

```
LB = LB + 1
```

```
GOTO 90
```

```
ENDIF
```

```
HEAD(PT) = 0
```

```
DO 120 CT = 1,ML-1,1
```

```
PT = MOD(PT+1,ML)
```

```
IF(HEAD(PT).NE.0) THEN
```

```
LAST(HEAD(PT)) = I
```

```
NEXT(I) = HEAD(PT)
```

```
I = NEXT(I)
```

```

      LB = LB + 1
      GOTO 90
    ENDIF
120 CONTINUE
C
C UPDATING THE NODES NODE POTENTIALS
C
130 DO 140 J=1, LB
      IJ = LABEL(J)
      R(IJ) = R(IJ) + D(IJ) - L
      K(A(IJ)) = K(A(IJ)) + L - D(IJ)
      IF(R(IJ).LT.KK) KK = R(IJ)
140 CONTINUE
      ML = CP - KK + 1
      I = IV
      AJ(A(I)) = AJ(A(I)) - 1
      AJ(A(RN)) = 1
C
C REVERSING THE ASSIGNMENTS ON THE SHORTESTR PATH TO THE FIRST
C ABUNDANT NODE
C
150 IF(I.NE.N+1) THEN
      A(I) = A(P(I))
      I = P(I)
      GOTO 150
    ENDIF
      DF = DF + 1
      GOTO 60
160 T1 = CTIME( )
      T10 = T1 - T0
      IF (.NOT.SWAP) THEN
        WRITE(6,190)' THE ASSIGNMENT PROBLEM IS INFEASIBLE !!!'
        GOTO 260
      ENDIF
C
C PRINTING THE OPTIMAL ASSIGNMENTS
C
      WRITE(6,180)'*** OPTIMAL SOLUTION REACHED ***'
      WRITE(4,210)'      TITLE : ',TITLE
      WRITE(4,210)'      SOURCE : ',FROM
      WRITE(4,210)'DESTINATION : ',TO
      WRITE(4,220)
      WRITE(4,230)
      WRITE(4,240)
      TOTC = 0
      DO 170 I=1,N
        DO 170 J=PN(I),PN(I+1)-1,1
          IF(TN(J).EQ.A(I)) WRITE(4,200)I,A(I),AL(J)
          IF(TN(J).EQ.A(I)) TOTC = TOTC + AL(J)
170 CONTINUE
      WRITE(4,200)
      WRITE(4,250)TOTC
      WRITE(4,*)'CPU TIME (SECONDS) : ',T10
180 FORMAT(///5X,A,///10X,'STOP..PROGRAM IS TERMINATED...')
190 FORMAT(15X,'| ',13,' | ',13,' | ',15,' | ')

```

```
200 FORMAT(15X,'-----')
210 FORMAT(/5X,A14,A)
220 FORMAT(///14X,'**** OPTIMAL ASSIGNMENTS ****',//15X,
+      '-----')
230 FORMAT(15X,' | FROM | TO | COST |')
240 FORMAT(15X,' |-----|')
250 FORMAT(15X,'      TOTAL COST | ',16,' | ',/31X,'-----')
260 STOP
    END
```

Appendix C

```
      A(I) = J
      AJ(J) = AJ(J) + 1
      HOLD = .FALSE.
    ENDIF
30  CONTINUE
    RN = N + 1
    CP = ML
    ML = CP + 1
    DK = 1
C
C: CHOOSEING THE DEFICIENT NODES
C
    DO 40 I=1,N
      IF(AJ(I).EQ.0) THEN
        DEFI(DK) = I
        DK = DK + 1
      ENDIF
40  CONTINUE
    DK = DK - 1
    IF(DK.EQ.0) GOTO 250
    DF = 1
    SWAP = .TRUE.
C
C: SOLVING THE ASSIGNMENT PROBLEM
C
60  IF(DF.GT.DK) GOTO 250
    A(RN) = DEFI(DF)
C
C: INITIALLIZATION
C
    DO 70 I=1,N
      D(I) = 99999
      LIST(I) = 0
      ARC(I) = 1
70  CONTINUE
    DO 80 I=1,LA
      NXT(I) = 0
80  CONTINUE
    DO 90 I=0,ML
      NEXT(I) = .FALSE.
      LAST(I) = 1
      FST(I) = LAST(I)
      IF(B(I).EQ.0) GOTO 90
      B(I) = 0
90  CONTINUE
    D(RN) = 0
    P(RN) = 0
    I = RN
    ARC(I) = 1
    SWAP = .FALSE.
    BI(I) = B(I)
    LB = 0
    L = 0
    IV = 0
    LOC = 0
```

```

LIST(1) = 0
IL = 0
100 IF(1.EQ.0) GOTO 250
IF(1.NE.RN) LABEL(LB) = 1
IF(1.NE.RN.AND.AJ(A(1)).GT.1) THEN
  SWAP = .TRUE.
  IV = 1
  L = D(1)
  GOTO 210
ENDIF
IC = 2
IL = IL + 1
LIST(IL) = 1
C
C SHORTEST PATH ALGORITHM (DANTZIG ADDRESS CALCULATION SORT)
C (LABEL SETTING ALGORITHM)
C
DO 130 IN=IL,1,-1
  I = LIST(IN)
  IF(1.EQ.0) GOTO 130
  MIN = 99999
  IF(ARC(I).GT.N) GOTO 130
  DO 120 NP=1,N
    IF(1.EQ.NP) GOTO 110
    DO 110 J=PN(NP),PN(NP+1)-1,1
      IF(TN(J).NE.A(I)) GOTO 110
      COST = AL(J) - R(NP) - K(A(I))
      IF(COST.GE.MIN) GOTO 110
      V = D(I) + COST
      IF(V.GE.D(J)) GOTO 110
      MIN = COST
      NODE = NP
110 CONTINUE
120 CONTINUE
  J = NODE
  V = D(I) + MIN
  ARC(I) = ARC(I) + 1
  IF(V.GE.D(J)) GOTO 130
  P(J) = I
  D(J) = V
  Q = MOD(V,ML)
  IQ = Q
  IF(LAST(Q).GT.1) NEXT(Q) = .TRUE.
  IF(LAST(Q).GT.N+1) NEXT(Q) = .FALSE.
  IF(LAST(Q).EQ.1) THEN
    B(Q) = REAL(J) + REAL(P(J)) / 1000.0
  ELSE
    NXT(J) = REAL(J) + REAL(P(J)) / 1000.0
  ENDIF
  LAST(Q) = LAST(Q) + 1
130 CONTINUE
140 LOC = 1
IF(B(IQ).EQ.0) GOTO 150
IF(NEXT(LOC)) THEN
  F = NXT(IC)

```

```

      NXT(IC) = 0
      GOTO 190
    ENDIF
150  LOC = I
      F = B(IQ)
      B(IQ) = 0
190  F4 = (F - REAL(INT(F))) * 1000.0 + 1.0
      I = INT(F)
      IP = INT(F4)
      LAST(LOC) = LAST(LOC) - 1
      IF(LAST(LOC).EQ.1) NEXT(LOC) = .FALSE.
      IF(IP.EQ.P(I)) THEN
        I = NXT(LOC+1)
        GOTO 140
      ENDIF
      LB = LB + 1
      GOTO 120
C
C UPDATING THE NODES NODE POTENTIALS
C
210  DO 220 J=1, LB
      IJ = LABEL(J)
      R(IJ) = R(IJ) + D(IJ) - L
      K(A(IJ)) = K(A(IJ)) + L - D(IJ)
      IF(R(IJ).LT.KK) KK = R(IJ)
220  CONTINUE
      ML = CP - KK + 1
      I = IV
      AJ(A(I)) = AJ(A(I)) - 1
      AJ(A(RN)) = 1
C
C REVERSING THE ASSIGNMENTS ON THE SHORTESTR PATH TO THE FIRST
C ABUNDANT NODE
C
230  IF(I.NE.N+1) THEN
      A(I) = A(P(I))
      I = P(I)
      GOTO 230
    ENDIF
      DF = DF + 1
      GOTO 60
250  T1 = CTIME()
      T10 = T1 - T0
      IF (.NOT.SWAP) THEN
        WRITE(6,270)' THE ASSIGNMENT PROBLEM IS INFEASIBLE !!!'
        GOTO 350
      ENDIF
C
C PRINTING THE OPTIMAL ASSIGNMENTS
C
      WRITE(6,270)'*** OPTIMAL SOLUTION REACHED ***'
      WRITE(4,300)'      TITLE : ',TITLE
      WRITE(4,300)'      SOURCE : ',FROM
      WRITE(4,300)'DESTINATION : ',TO
      WRITE(4,310)

```

```
WRITE(4,320)
WRITE(4,330)
TOTC = 0
DO 260 I=1,N
DO 260 J=PN(I),PN(I+1)-1,1
IF(TN(J) .EQ. A(I)) WRITE(4,280)I,A(I),AL(J)
IF(TN(J) .EQ. A(I)) TOTC = TOTC + AL(J)
260 CONTINUE
WRITE(4,290)
WRITE(4,340)TOTC
WRITE(4,*)'CPU - TIME (SECONDS) :',T10
270 FORMAT(///5X,A,//10X,'STOP.. PROGRAM IS TERMINATED...')
280 FORMAT(15X,'| ',13,' | ',13,' | ',15,' |')
290 FORMAT(15X,'-----')
300 FORMAT(/5X,A14,A)
310 FORMAT(///14X,'**** OPTIMAL ASSIGNMENTS ****',//15X,
+ '-----')
320 FORMAT(15X,'| FROM | TO | COST |')
330 FORMAT(15X,'|-----|')
340 FORMAT(15X,' TOTAL COST | ',16,' |',/31X,'-----')
350 STOP
END
```


Appendix D

C
 C THIS PROGRAM SOLVE THE ASSIGNMENT PROBLEM AS A SUCCESSIVE
 C SHORTEST PATH ALGORITHM USING PAPE ALGORITHM TO SOLVE THE
 C SHORTEST PATH PROBLEM.

C
 PROGRAM SPAN_III
 PARAMETER (N=100,LA=10000)
 INTEGER PN(N+1),A(N+1),AJ(N),R(N),K(N),TN(LA),CP
 INTEGER RN,P(N+1),V,IV,DF,DK,RST(N),PV,L,SCAN(N+1)
 INTEGER AL(LA),TOTC,D(N+1),DEFI(N),B(LA),H(N+1),COST
 LOGICAL SWAP,HOLD
 CHARACTER FROM*10, TO*10, TITLE*35
 EXTERNAL CTIME
 DATA (AJ(I),I=1,N)/N*0/
 DATA (K(I),I=1,N)/N*0/
 READ(1,*)FROM,TO
 TITLE = FROM
 L1 = INDEX(TITLE, ' ')
 L2 = INDEX(TO, ' ')
 TITLE(L1+1:L1+L2) = TO
 L3 = L1 + L2 + 1
 TITLE(L3:) = 'ASSIGNMENTS'
 II = 0
 ML = -99999
 KK = 99999
 PN(1) = 1

C
 C IN THIS SECTION THE DATA HAS TO BE READ AS :
 C FROM NODE, TO NODE, AND COST

C
 10 READ(1,*,END=20) I,J,COST
 IM = IM + 1
 TN(IM) = J
 AL(IM) = COST
 IF(II.NE.1) R(1) = 999999
 IF(II.NE.1) PN(I+1) = PN(1)
 PN(I+1) = PN(I+1) + 1
 IF(COST.LT.R(1)) R(1) = COST
 IF(COST.GT.ML) ML = COST
 II = I
 GOTO 10

C
 C BRINGING THE PROBLEM INTO THE STANDARD FORM

C
 20 TO = CTIME()
 DO 40 I=1,N
 HOLD = .TRUE.
 DO 30 JK=PN(I),PN(I+1)-1,1
 J = TN(JK)
 COST = AL(JK) - R(1)
 IF(COST.EQ.0.AND.HOLD) THEN
 A(I) = J
 AJ(J) = AJ(J) + 1
 HOLD = .FALSE.
 ENDDO
 ENDDO

```
30 CONTINUE
40 CONTINUE
   RN = N + 1
   CP = ML
   ML = CP + 1
```

```
C
C CHOOSING THE DEFICIENT NODES
C
```

```
   DK = 1
   DO 50 I=1,N
     IF(AJ(I).EQ.0) THEN
       DEFI(DK) = I
       DK = DK + 1
     ENDIF
50 CONTINUE
   DK = DK - 1
   DF = 1
   IF(DK.EQ.0) GOTO 180
```

```
C
C SOLVING THE ASSIGNMENT PROBLEM
```

```
C
60 IF(DF.GT.DK) GOTO 180
   A(RN) = DEFI(DF)
```

```
C
C INITIALLIZATION
```

```
C
   DO 70 I=1,N
     D(I) = 99999
     P(I) = 0
     H(I) = 0
     RST(I) = 0
     SCAN(I) = 0
70 CONTINUE
   D(RN) = 0
   P(RN) = 0
   DO 90 I=1,N
     DO 80 J1=PN(I),PN(I+1)-1,1
       J = TN(J1)
       IF(J.EQ.A(I)) THEN
         RST(J) = RST(J) + 1
       ENDIF
80 CONTINUE
90 CONTINUE
   I = RN
   SCAN(I) = 0
   LOC = 0
   IL = 1
   IV = 0
   L = 99999
   SWAP = .FALSE.
```

```
C
```

C SHORTEST PATH ALGORITHM (PAPE ALGORITHM)

C (LABEL CORRECTING ALGORITHM)

C

```

100 DO 145 NP=1,N,1
    DO 140 K1=PN(NP),PN(NP+1)-1,1
        IF(TN(K1).NE.A(1)) GOTO 140
        J = NP
        IF(SCAN(1).EQ.0)RST(J) = RST(J) - 1
        COST = AL(J) - R(NP) - K(A(1))
        V = D(1) + COST
        IF (V.GE.D(J)) GOTO 140
        P(J) = 1
        D(J) = V
        IF(H(J).EQ.1.OR.AJ(A(J)).GT.1) GO TO 110
        LOC = LOC + 1
        B(LOC) = J
        H(J) = 1
110 IF(AJ(A(J)).GT.1) THEN
        IF(D(J).LT.L) THEN
            SWAP = .TRUE.
            L = D(J)
            IV = J
            IF(RST(J).NE.0) GOTO 140
            PV = P(IV)
120 IF(PV.NE.RN) THEN
            IF(RST(PV).NE.0) GOTO 140
            PV = P(PV)
            GO TO 120
        ENDIF
        DO 130 IJ = IL,LOC,1
            IF(D(B(IJ)).LT.L.OR.RST(B(IJ)).GT.0) GOTO 140
130 CONTINUE
        GOTO 150
    ENDIF
    ENDIF
140 CONTINUE
    IF(SCAN(1).EQ.0) SCAN(1) = 1
    IF(IL.GT.LOC) GOTO 150
    I = B(IL)
    H(I) = 0
    IL = IL + 1
145 CONTINUE

```

C

C UPDATING THE NODES NODE POTENTIALS

C

```

150 IF(.NOT.SWAP) GOTO 180
    DO 160 J=1,N
        IF(RST(J).EQ.0.AND.D(J).LT.L) THEN
            R(J) = R(J) + D(J) - L
            K(A(J)) = K(A(J)) + L - D(J)
            IF(R(J).LT.KK) KK = R(J)
        ENDIF
160 CONTINUE
    ML = CP - KK + 1
    AJ(A(RN)) = 1

```

```

      I = IV
      AJ(A(I)) = AJ(A(I)) - 1
C
C REVERSING THE ASSIGNMENTS ON THE SHORTESTR PATH TO THE FIRST
C ABUNDANT NODE
C
170  IF(I.NE.RN) THEN
      A(I) = A(P(I))
      I = P(I)
      GOTO 170
    ENDIF
    DF = DF + 1
    GOTO 60
180  T1 = CTIME()
      T10 = T1 - T0
      IF (.NOT.SWAP) THEN
        WRITE(6,200)' THE ASSIGNMENT PROBLEM IS INFEASIBLE !!!'
        GOTO 280
      ENDIF
C
C PRINTING THE OPTIMAL ASSIGNMENTS
C
      WRITE(6,200)'*** OPTIMAL SOLUTION REACHED ***'
      WRITE(4,220)'      TITLE : ',TITLE
      WRITE(4,220)'      SOURCE : ',FROM
      WRITE(4,220)'DESTINATION : ',TO
      WRITE(4,230)
      WRITE(4,240)
      WRITE(4,250)
      TOTC = 0
      DO 190 I=1,N
        DO 190 J=PN(I),PN(I+1)-1,1
          IF(TN(J).EQ.A(I)) WRITE(4,210)I,A(I),AL(J)
          IF(TN(J).EQ.A(I)) TOTC = TOTC + AL(J)
190  CONTINUE
      WRITE(4,270)
      WRITE(4,260)TOTC
      WRITE(4,*)'CPU - TIME (SECONDS) :',T10
200  FORMAT(///5X,A,///10X,'STOP..PROGRAM IS TERMINATED...')
210  FORMAT(15X,' | ',13,' | ',13,' | ',15,' | ')
220  FORMAT(/5X,A14,A)
230  FORMAT(///14X,'**** OPTIMAL ASSIGNMENTS ****',//15X,
+      '-----')
240  FORMAT(15X,' | FROM | TO | COST |')
250  FORMAT(15X,' |-----|')
260  FORMAT(15X,'      TOTAL COST | ',16,' | ',/31X,'-----')
270  FORMAT(15X,'-----')
280  STOP
      END

```

Appendix E

C
 C THIS PROGRAM SOLVE THE SEMI-ASSIGNMENT PROBLEM AS A SUCCESSIVE
 C SHORTEST PATH ALGORITHM USING DIJKSTRA ALGORITHM TO SOLVE THE
 C SHORTEST PATH PROBLEM.

C
 PROGRAM SPSN
 PARAMETER (N=50,M=500,LA=2000,IH=1000,NM=LA)
 INTEGER PN(N+1),A(LA),R(N),K(M),BJ(M),T(LA),TN(LA)
 INTEGER RN,P(N+1),V,L,PT,CT,Q,IV,DF,DK,LABEL(N),LB
 INTEGER AL(LA),TOTC,D(N+1),DEFI(M),COST,CP,SV(N),PV(N)
 INTEGER NEXT(0:N+1),LAST(0:N+1)
 INTEGER HEAD(0:IH),TAIL(0:IH)
 LOGICAL SWAP,HOLD
 EXTERNAL CTIME
 DATA (A(I),I=1,LA)/NM*0/
 DATA (K(I),I=1,M)/M*0/
 DATA (BJ(I),I=1,M)/M*0/
 II = 0
 ML = -999
 KK = 99999
 PN(1) = 1
 IM = 0

C
 C READ THE TOTAL NUMBER OF ASSIGNMENT FOR EACH SOURCE NODE.

C
 DO 10 I=1,N
 10 READ(5,*)P(I)

C
 C IN THIS SECTION THE DATA HAS TO BE READ AS :
 C FROM NODE, TO NODE, AND COST

C
 20 READ(5,*,END=30) I,J,COST
 IM = IM + 1
 TN(IM) = J
 AL(IM) = COST
 IF(II.NE.I) R(I) = 999999
 IF(II.NE.I) PN(I+1) = PN(I)
 PN(I+1) = PN(I+1) + 1
 IF(COST.LT.R(I)) R(I) = COST
 IF(COST.GT.KK) KK = COST
 IF(COST.GT.ML) ML = COST
 II = I
 GO TO 20

C
 C BRINGING THE PROBLEM INTO THE STANDARD FORM

C
 30 T0 = CTIME()
 DO 50 I=1,N
 HOLD = .TRUE.
 DO 40 JK=PN(I),PN(I+1)-1,1
 J = TN(JK)
 COST = AL(JK) - R(I)
 IF(COST.EQ.0.AND.HOLD) THEN
 BJ(J) = BJ(J) + P(I)
 A(JK) = J

```
        T(JK) = P(1)
        HOLD = .FALSE.
    ENDIF
40  CONTINUE
    P(1) = 0
50  CONTINUE
    RN = N + 1
    CP = ML
    ML = CP + 1
    DK = 1
C
C CHOOSEING THE DEFICIENT NODES
C
    DO 60 J=1,M
        IF(BJ(J).EQ.0) THEN
            DEFI(DK) = J
            DK = DK + 1
        ENDIF
60  CONTINUE
    DK = DK - 1
    DF = 1
    SWAP = .TRUE.
C
C SOLVING THE SEMI-ASSIGNMENT PROBLEM
C
70  IF(DF.GT.DK) GOTO 180
C
C INITIALIZATION
C
    DO 80 I=1,ML-1
        HEAD(I) = 0
        TAIL(I) = 0
80  CONTINUE
    DO 90 I=1,N
        D(I) = 99999
        P(I) = 0
90  CONTINUE
    PT = 0
    D(RN) = 0
    NEXT(RN) = 0
    LAST(RN) = 0
    P(RN) = 0
    HEAD(0) = RN
    I = HEAD(0)
    TAIL(0) = RN
    SWAP = .FALSE.
    LB = 0
    L = 0
    IV = 0
    IC = 0
100 IF(I.EQ.RN) IR = DEFI(DF)
    IF(I.NE.RN) IR = A(PN(I)+IC)
    IF(I.EQ.0) GOTO 180
    IF(I.NE.RN) LABEL(LB) = I
    IF(I.NE.RN.AND.BJ(IR).GT.1) THEN
```



```

SWAP = .TRUE.
IV = I
IVC = IR
IAC = PN(I) + IC
L = D(I)
GOTO 140
ENDIF

```

C

C SHORTEST PATH ALGORITHM (DIJKSTRA ADDRESS CALCULATION SORT)

C (LABEL SETTING ALGORITHM)

C

```

DO 120 NP=1,N,1
  IF(I.EQ.NP) GO TO 120
DO 110 K1=PN(NP),PN(NP+1)-1,1
  IF(TN(K1).NE.A(IR)) GO TO 110
  COST = AL(K1) - R(NP) - K(IR)
  V = D(I) + COST
  J = NP
  IF (V.LT.D(J)) THEN
    IF (D(J).NE.99999) THEN
      Q = MOD(D(J),ML)
      IF (HEAD(Q).EQ.J) THEN
        HEAD(Q) = NEXT(J)
      ELSE
        IF(TAIL(Q).EQ.J) THEN
          TAIL(Q) = LAST(J)
          NEXT(LAST(J)) = 0
        ELSE
          LAST(NEXT(J)) = LAST(J)
          NEXT(LAST(J)) = NEXT(J)
        ENDIF
      ENDIF
    ENDIF
    Q = MOD(V,ML)
    IF (HEAD(Q).EQ.0) THEN
      HEAD(Q) = J
      LAST(J) = 0
    ELSE
      LAST(J) = TAIL(Q)
      NEXT(TAIL(Q)) = J
    ENDIF
    P(J) = I
    D(J) = V
    SV(NP) = K1
    PV(NP) = IR
    TAIL(Q) = J
    NEXT(J) = 0
  ENDIF
110 CONTINUE
120 CONTINUE
  IC = IC + 1
  IR = A(PN(I)+IC)
  IF(IR.EQ.TN(PN(I)+IC)) GOTO 100
  IF (NEXT(I).NE.0) THEN
    I = NEXT(I)

```

```

    LB = LB + 1
    IC = 0
    GOTO 100
ENDIF
HEAD(PT) = 0
DO 130 CT = 1,ML-1,1
  PT = MOD(PT+1,ML)
  IF(HEAD(PT).NE.0) THEN
    LAST(HEAD(PT)) = 1
    NEXT(I) = HEAD(PT)
    I = NEXT(I)
    LB = LB + 1
    IC = 1
    GOTO 100
  ENDIF
130 CONTINUE
C
C UPDATING THE NODES NODE POTENTIALS
C
140 DO 160 J=1,LB
  IJ = LABEL(J)
  R(IJ) = R(IJ) + D(IJ) - L
  DO 150 IC = PN(IJ),PN(IJ+1)-1,1
    IF(R(IJ) .LT. KK) KK = R(IJ)
    IR = A(IC)
    IF(IR.EQ.0) GOTO 160
    K(IR) = K(IR) + L - D(IJ)
  150 CONTINUE
160 CONTINUE
C
C REVERSRING THE ASSIGNMENTS ON THE SHORTESTR PATH TO THE FIRST
C ABUNDANT NODE
C
  BJ(IVC)      = BJ(IVC) - 1
  BJ(DEFI(DF)) = 1
  I = IV
170 IF ( I .NE. N+1 ) THEN
  IF(T(SV(I)) .EQ. 1) THEN
    A(SV(I)) = PV(I)
    A(IAC)   = 0
    T(SV(I)) = 1
    T(IAC)   = 0
  ELSE
    A(SV(I)) = PV(I)
    A(IAC)   = 0
    T(SV(I)) = 1
    T(IAC)   = T(IAC) - 1
  ENDIF
  IAC = SV(I)
  I   = P(I)
  IC = IC + 1
GOTO 170
ENDIF
DF = DF + 1
ML = CP - KK + 1

```

```
GOTO 70
180 IF (.NOT.SWAP) THEN
  WRITE(6,210)' THE SEMI-ASSIGNMENT PROBLEM IS INFEASIBLE !!!'
  GOTO 290
ENDIF
T1 = CTIME()
T10= T1 - T0
C
C PRINTING THE OPTIMAL ASSIGNMENTS
C
  WRITE(6,210)'*** OPTIMAL SOLUTION REACHED ***'
  WRITE(6,240)
  WRITE(6,250)
  WRITE(6,260)
  TOTC = 0
  DO 200 I=1,N
  DO 190 J=PN(I),PN(I+1)-1,1
  IF (A(J).EQ.0) GOTO 200
  IF (IN(J).EQ.A(J)) WRITE(6,220)I,TN(J),AL(J),1
  IF (IN(J).EQ.A(J)) TOTC = TOTC + AL(J)
190 CONTINUE
200 CONTINUE
  WRITE(6,230)
  WRITE(6,270)TOTC
  WRITE(6,280)
  WRITE(6,*) 'CPU TIME (SECONDS) :',T10
210 FORMAT(/17X,A,/29X,'FOR THE',/22X,'SEMI-ASSIGNMENT PROBLEM ')
220 FORMAT(15X,'| ',13,' | ',13,' | ',15,' | ',15,' | ')
230 FORMAT(15X,'-----')
240 FORMAT(/15X,'-----')
250 FORMAT(15X,'| FROM | TO | COST | SHIPMENT |')
260 FORMAT(15X,'|-----|')
270 FORMAT(/15X,' TOTAL SEMI-ASSIGNMENT COST = ',16)
280 FORMAT(/21X,' *** END OF OUTPUT ***')
290 STOP
  END
```

C
 C THIS PROGRAM SOLVE THE ASSIGNMENT PROBLEM AS A SUCCESSIVE
 C SHORTEST PATH ALGORITHM USING DANTZIG ALGORITHM TO SOLVE
 C THE SHORTEST PATH PROBLEM.

C

```

PROGRAM SPAN_11
PARAMETER (N=100, LA=10000, IH=100)
INTEGER PN(N+1), A(N+1), AJ(N), R(N), K(N), ARC(N+1)
INTEGER RN, P(N+1), V, L, IV, DF, DK, LABEL(N), LB, Q, COST
INTEGER AL(LA), TOTC, D(N+1), DEFI(N), LIST(N+1), CP
INTEGER LAST(0:N), TN(LA), BI(0: IH), FST(N)
REAL B(0: IH), NXT(N)
LOGICAL SWAP, HOLD, NEXT(0: LA)
CHARACTER FROM*10, TO*10, TITLE*35
EXTERNAL CTIME
DATA (K(I), I=1, N)/N*0/
DATA (AJ(I), I=1, N)/N*0/
READ(1, *) FROM, TO
TITLE = FROM
L1 = INDEX(TITLE, ' ')
L2 = INDEX(TO, ' ')
TITLE(L1+1: L1+L2) = TO
L3 = L1 + L2 + 1
TITLE(L3:) = 'ASSIGNMENTS'
II = 0
ML = -999
KK = 99999
IM = 0
PN(1) = 1

```

C

C IN THIS SECTION THE DATA HAS TO BE READ AS :
 C FROM NODE, TO NODE, AND COST

C

```

10 READ(1, *, END=20) I, J, COST
IM = IM + 1
TN(IM) = J
AL(IM) = COST
IF(II.NE.1) R(I) = 999999
IF(II.NE.1) PN(I+1) = PN(I)
PN(I+1) = PN(I+1) + 1
IF(COST.LT.R(I)) R(I) = COST
IF(COST.LT.KK) KK = COST
IF(COST.GT.ML) ML = COST
II = I
GOTO 10

```

C

C BRINGING THE PROBLEM INTO THE STANDARD FORM

C

```

20 TO = CTIME()
DO 30 I=1, N
HOLD = .TRUE.
DO 30 JK=PN(I), PN(I+1)-1, 1
J = TN(JK)
COST = AL(JK) - R(I)
IF(COST.EQ.0.AND.HOLD) THEN

```

Appendix F

C
 C THIS PROGRAM SOLVE THE SEMI-ASSIGNMENT PROBLEM AS AN
 C ASSIGNMENT PROBLEM . THE SEMI-ASSIGNMENT PROBLEM
 C IS CONVERTED TO AN ASSIGNMENT PROBLEM . THEN THE
 C THE NEW ASSIGNMENTPROBLEM WILL BE SOLVED AS A SSP
 C ALGORITHM . DIJKSTRA ALGORITHM IS USED TO SOLVE THE
 C SHORTEST PATH PROBLEM.

C

```

PROGRAM SPSAN
PARAMETER (NS=50,M=500,LA=2000,IH=1000)
INTEGER PN(M+1),A(M+1),AJ(M),R(M),K(M),TN(LA)
INTEGER RN,P(M+1),V,L,PT,CT,Q,IV,DF,DK,LABEL(M),LB
INTEGER AL(LA),TOTC,D(M+1),DEFI(M),COST,CP
INTEGER NEXT(M+1),LAST(M+1)
INTEGER HEAD(0:IH),TAIL(0:IH)
LOGICAL SWAP,HOLD
CHARACTER FROM*10, TO*10, TITLE*35
EXTERNAL CTIME
DATA (K(I),I=1,M)/M*0/
DATA (AJ(I),I=1,M)/M*0/
READ(1,*)FROM,TO
TITLE = FROM
L1 = INDEX(TITLE,' ')
L2 = INDEX(TO,' ')
TITLE(L1+1:L1+L2) = TO
L3 = L1 + L2 + 1
TITLE(L3:) = 'ASSIGNMENTS'

```

C

C READ THE TOTAL NUMBER OF ASSIGNMENTS
 C THAT EACH SOURCE NODE CAN GIVE.

C

```

DO 5 I=1,NS
  READ(1,*) P(I)
5 CONTINUE
II = 0
ML = -999
KK = 99999
PN(1) = 1
IM = 0

```

C

C IN THIS SECTION THE DATA HAS TO BE READ AS :
 C FROM NODE, TO NODE, AND COST

C

```

10 READ(1,*,END=20) I,J,COST
  IM = IM + 1
  TN(IM) = J
  AL(IM) = COST
  IF(II.NE.I) R(I) = 999999
  IF(II.NE.I) PN(I+1) = PN(I)
  PN(I+1) = PN(I+1) + 1
  IF(COST.LT.R(I)) R(I) = COST
  IF(COST.LT.KK) KK = COST
  IF(COST.GT.ML) ML = COST
  II = I
GOTO 10

```

```
C
C TRANSFER THE PROBLEM FROM SEMI-ASSIGNMENT
C TO ASSIGNMENT PROBLEM
C
20 TO = CTIME()
NEW = NS
NP = NS + 1
LT = IM
DO 26 I=1,NS
22 IF(P(I).EQ.1) GOTO 26
P(I) = P(I) - 1
NEW = NEW + 1
R(NEW) = R(I)
PN(NP+1) = PN(NP)
DO 24 J=PN(I),PN(I+1)-1,1
PN(NP+1) = PN(NP+1) + 1
LT = LT + 1
TN(LT) = TN(J)
AL(LT) = 99999
24 CONTINUE
GO TO 22
26 CONTINUE
C
C BRINGING THE PROBLEM INTO THE STANDARD FORM
C
N = M
DO 40 I=1,N
HOLD = .TRUE.
DO 30 JK=PN(I),PN(I+1)-1,1
J = TN(JK)
COST = AL(JK) - R(I)
IF(COST.EQ.0.AND.HOLD) THEN
A(I) = J
AJ(J) = AJ(J) + 1
HOLD = .FALSE.
ENDIF
30 CONTINUE
40 CONTINUE
RN = N + 1
CP = ML
ML = CP + 1
DK = 1
C
C CHOOSEING THE DEFICIENT NODES
C
DO 50 I=1,N
IF(AJ(I).EQ.0) THEN
DEFI(DK) = I
DK = DK + 1
ENDIF
50 CONTINUE
DK = DK - 1
DF = 1
SWAP = .TRUE.
C
```

C SOLVING THE ASSIGNMENT PROBLEM

C

60 IF(DF.GT.DK.OR.(.NOT.SWAP)) GOTO 160

C

C INITIALLIZATION

C

A(RN) = DEFI(DF)

DO 70 I=1,ML-1

HEAD(I) = 0

TAIL(I) = 0

70 CONTINUE

DO 80 I=1,N

D(I) = 99999

80 CONTINUE

PT = 0

D(RN) = 0

NEXT(RN) = 0

LAST(RN) = 0

P(RN) = 0

HEAD(0) = RN

I = HEAD(0)

TAIL(0) = RN

SWAP = .FALSE.

LB = 0

L = 0

IV = 0

90 IF(I.EQ.0) GOTO 160

IF(I.NE.RN) LABEL(LB) = I

IF(I.NE.RN.AND.AJ(A(I)).GT.1) THEN

SWAP = .TRUE.

IV = I

L = D(I)

GOTO 130

ENDIF

C

C SHORTEST PATH ALGORITHM (DIJKSTRA ADDRESS CALCULATION SORT)

C (LABEL SETTING ALGORITHM)

C

DO 110 NP=1,N

IF(I.EQ.NP) GOTO 110

DO 100 K1=PN(NP),PN(NP+1)-1,1

IF(TN(K1).NE.A(I)) GOTO 100

COST = AL(K1) - R(NP) - K(A(I))

V = D(I) + COST

J = NP

IF (V.LT.D(J)) THEN

IF (D(J).NE.99999) THEN

Q = MOD(D(J),ML)

IF (HEAD(Q).EQ.J) THEN

HEAD(Q) = NEXT(J)

ELSE

IF(TAIL(Q).EQ.J) THEN

TAIL(Q) = LAST(J)

NEXT(LAST(J)) = 0

ELSE


```

        LAST(NEXT(J)) = LAST(J)
        NEXT(LAST(J)) = NEXT(J)
    ENDIF
ENDIF
ENDIF
Q = MOD(V,ML)
IF (HEAD(Q).EQ.0) THEN
    HEAD(Q) = J
    LAST(J) = 0
ELSE
    LAST(J) = TAIL(Q)
    NEXT(TAIL(Q)) = J
ENDIF
P(J) = 1
D(J) = V
TAIL(Q) = J
NEXT(J) = 0
ENDIF
100 CONTINUE
110 CONTINUE
IF (NEXT(I).NE.0) THEN
    I = NEXT(I)
    LB = LB + 1
    GOTO 90
ENDIF
HEAD(PT) = 0
DO 120 CT = 1,ML-1,1
    PT = MOD(PT+1,ML)
    IF(HEAD(PT).NE.0) THEN
        LAST(HEAD(PT)) = I
        NEXT(I) = HEAD(PT)
        I = NEXT(I)
        LB = LB + 1
        GOTO 90
    ENDIF
120 CONTINUE
C
C UPDATING THE NODES NODE POTENTIALS
C
130 DO 140 J=1, LB
    IJ = LABEL(J)
    R(IJ) = R(IJ) + D(IJ) - L
    K(A(IJ)) = K(A(IJ)) + L - D(IJ)
    IF(R(IJ).LT.KK) KK = R(IJ)
140 CONTINUE
ML = CP - KK + 1
I = IV
AJ(A(I)) = AJ(A(I)) - 1
AJ(A(RN)) = 1
C
C REVERSING THE ASSIGNMENTS ON THE SHORTESTR PATH TO THE FIRST
C ABUNDANT NODE
C
150 IF(I.NE.N+1) THEN
    A(I) = A(P(I))

```

```

      I = P(I)
      GOTO 150
    ENDIF
    DF = DF + 1
    GOTO 60
160  T1 = CTIME()
      T10 = T1 - T0
      IF (.NOT.SWAP) THEN
        WRITE(6,190)' THE ASSIGNMENT PROBLEM IS INFEASIBLE !!!'
        GOTO 260
      ENDIF
C
C PRINTING THE OPTIMAL ASSIGNMENTS
C
      WRITE(6,180)'*** OPTIMAL SOLUTION REACHED ***'
      WRITE(4,210)'      TITLE : ',TITLE
      WRITE(4,210)'      SOURCE : ',FROM
      WRITE(4,210)'DESTINATION : ',TO
      WRITE(4,220)
      WRITE(4,230)
      WRITE(4,240)
      TOTC = 0
      DO 170 I=1,N
      DO 170 J=PN(I),PN(I+1)-1,1
      IF(IN(J).EQ.A(I)) WRITE(4,200)I,A(I),AL(J)
      IF(TN(J).EQ.A(I)) TOTC = TOTC + AL(J)
170  CONTINUE
      WRITE(4,200)
      WRITE(4,250)TOTC
      WRITE(4,*)'CPU TIME (SECONDS) : ',T10
180  FORMAT(///5X,A,///10X,'STOP..PROGRAM IS TERMINATED...')
190  FORMAT(15X,'| ',13,' | ',13,' | ',15,' |')
200  FORMAT(15X,'-----')
210  FORMAT(/5X,A14,A)
220  FORMAT(///14X,'**** OPTIMAL ASSIGNMENTS ****',//15X,
+      '-----')
230  FORMAT(15X,'| FROM | TO | COST |')
240  FORMAT(15X,'|-----|')
250  FORMAT(15X,'      TOTAL COST | ',16,' |',/31X,'-----')
260  STOP
      END

```

Appendix G

C
 C THIS PROGRAM SOLVE THE TRANSPORTATION PROBLEM AS A SUCCESSIVE
 C SHORTEST PATH ALGORITHM USING DIJKSTRA ALGORITHM TO SOLVE THE
 C SHORTEST PATH PROBLEM.

C
 PROGRAM SPTN
 PARAMETER (N=150,M=150,LA=3000,IH=20000)
 INTEGER PN(M),A(M+1),R(N),K(M),BJ(M),T(LA),H(LA),TN(LA)
 INTEGER RN,P(M+1),V,L,PT,CT,Q,IV,DF,DK,LABEL(M),LB
 INTEGER AL(LA),TOTC,D(N+1),DEFI(M),COST,DEM(M),CP
 INTEGER NEXT(N+1),LAST(N+1),TRNS(M),ORG(M)
 INTEGER HEAD(0:IH),TAIL(0:IH)
 LOGICAL SWAP,HOLD
 EXTERNAL CTIME
 DATA (PN(I),I=1,N)/N*0/
 DATA (A(I),I=1,M+1)/(M+1)*0/
 DATA (T(I),I=1,N)/LA*0/
 DATA (K(I),I=1,M)/M*0/
 DATA (BJ(I),I=1,M)/M*0/
 II = 0
 ML = -999
 KK = 99999
 DO 5 I=1,N
 READ(1,*)P(I)
 5 CONTINUE
 DO 6 I=1,M
 READ(1,*)DEM(I)
 ORG(I) = DEM(I)
 6 CONTINUE
 10 READ(1,*,END=20) I,J,COST
 IM = IM + 1
 TN(IM) = J
 AL(IM) = COST
 IF(II.NE.I) R(I) = 999999
 IF(COST.LT.R(I)) R(I) = COST
 IF(COST.LT.KK) KK = COST
 IF(COST.GT.ML) ML = COST
 II = I
 GOTO 10
 20 DO 30 I=1,N
 HOLD = .TRUE.
 DO 40 JK=PN(I),PN(I+1)-1,1
 J = TN(JK)
 COST = AL(JK) - R(I)
 IF(COST.EQ.0.AND.HOLD) THEN
 BJ(J) = P(I) + BJ(J)
 A(I) = J
 T(I) = P(I)
 HOLD = .FALSE.
 ENDIF
 40 CONTINUE
 P(I) = 0
 30 CONTINUE
 TO = CTIME()
 RN = N + 1

```
CP = ML
ML = CP + 1
DK = 1
DO 70 J=1,M
  IDIF = BJ(J) - ORG(J)
  IF(IDIF.LT.0) THEN
    DEFI(DK) = J
    DK = DK + 1
  ENDIF
70 CONTINUE
DK = DK - 1
DF = 1
SWAP = .TRUE.
80 IF(DF.GT.DK) GOTO 170
88 A(RN) = DEFI(DF)
DO 110 I=1,ML-1
  HEAD(I) = 0
  TAIL(I) = 0
110 CONTINUE
DO 57 I=1,N
  D(I) = 99999
57 CONTINUE
PT = 0
D(RN) = 0
NEXT(RN) = 0
LAST(RN) = 0
P(RN) = 0
HEAD(0) = RN
I = HEAD(0)
TAIL(0) = RN
SWAP = .FALSE.
LB = 0
L = 0
IV = 0
IC = 1
120 IF(I.EQ.0) GOTO 170
IF(I.NE.RN) LABEL(LB) = I
IF(I.NE.RN.AND.BJ(A(IC)).GT.ORG(A(IC))) THEN
  SWAP = .TRUE.
  IV = I
  IVC = IC
  L = D(I)
  GOTO 150
ENDIF
C
C SHORTEST PATH ALGORITHM (DIJKSTRA ADDRESS CALCULATION SORT)
C (LABEL-SETTING ALGORITHM.
C
DO 130 K1=1,N,1
  IF(I.EQ.K1) GOTO 130
  COST = AL(A(I)) - R(K1) - K(IC)
  V = D(I) + COST
  J = K1
  IF (V.LT.D(J)) THEN
    IF (D(J).NE.99999) THEN
```

```

      Q = MOD(D(J),ML)
      IF (HEAD(Q).EQ.J) THEN
        HEAD(Q) = NEXT(J)
      ELSE
        IF (TAIL(Q).EQ.J) THEN
          TAIL(Q) = LAST(J)
          NEXT(LAST(J)) = 0
        ELSE
          LAST(NEXT(J)) = LAST(J)
          NEXT(LAST(J)) = NEXT(J)
        ENDIF
      ENDIF
    ENDIF
  Q = MOD(V,ML)
  IF (HEAD(Q).EQ.0) THEN
    HEAD(Q) = J
    LAST(J) = 0
  ELSE
    LAST(J) = TAIL(Q)
    NEXT(TAIL(Q)) = J
  ENDIF
  P(J) = I
  D(J) = V
  TAIL(Q) = J
  NEXT(J) = 0
  ENDIF
130 CONTINUE
  IC = IC + 1
  IF(A(IC).NE.0) GOTO 120
  IF (NEXT(I).NE.0) THEN
    I = NEXT(I)
    LB = LB + 1
    IC = 1
    GOTO 120
  ENDIF
  HEAD(PT) = 0
  DO 140 CT = 1,ML-1,1
    PT = MOD(PT+1,ML)
    IF(HEAD(PT).NE.0) THEN
      LAST(HEAD(PT)) = I
      NEXT(I) = HEAD(PT)
      I = NEXT(I)
      LB = LB + 1
      IC = 1
      GOTO 120
    ENDIF
  CONTINUE
140 *****
150 DO 321 JK=1,M
321 H(JK) = 0
    DO 123 J=1,LB
      IC = 1
      IJ = LABEL(J)
      R(IJ) = R(IJ) + D(IJ) - L
122 IF(A(IC).EQ.0) GOTO 129

```

```

      IF(H(A(IJ)) .EQ . 1) GOTO 322
      K(A(IC)) = K(A(IC)) + L - D(IJ)
      H(A(IC)) = 1
322   IC = IC + 1
      GOTO 122
129   IF(R(IJ).LT.KK) KK=R(IJ)
123   CONTINUE
      ML = CP - KK + 1
      MSHIP = DEM(DEFI(DF)) - BJ(A(RN))
      I = IV
      IA = IV
      IC = 1
      ISHIP = MIN(BJ(A(IVC))-DEM(A(IVC)),T(IV))
      IF (MSHIP .LT. ISHIP) ISHIP = MSHIP
      I = P(IV)
161   IF ( I.NE. N+1) THEN
      IC = 1
      IM = 9999
162   IF(A(IC).NE.0) THEN
      IN = AL(IA) - R(IA) - K(IC)
      IF(IN.LT.IM) THEN
      IM = IN
      ISC = IC
      ENDIF
165   IC = IC + 1
      GOTO 162
      MSHIP = T(ISC)
      IF (MSHIP .LT. ISHIP) ISHIP = MSHIP
      IA = I
      I = P(I)
      GOTO 161
      ENDIF
c: *****
      BJ(A(IVC)) = BJ(A(IVC)) - ISHIP
      TRNS(A(IVC)) = BJ(A(IVC))
      BJ(A(RN)) = BJ(A(RN)) + ISHIP
      IA = IV
      IAC = IVC
      I = P(IA)
151   IF ( I.NE. 0 ) THEN
      IC = 1
      IM = 9999
152   IF(A(IC).NE.0) THEN
      IN = AL(IA) - R(IA) - K(IC)
      IF(IN.LT.IM) THEN
      IM = IN
      ISC = IC
      ENDIF
155   IC = IC + 1
      GOTO 152
      ENDIF
      IC = ISC
      I4 = T(IAC)
      IF(TRNS(IC).NE.0) THEN
      T(IC) = T(IC) + ISHIP

```

```
T(IAC) = T(IAC) - ISHIP
BJ(IC) = BJ(IC) + ISHIP
IF( I4 .NE. ISHIP ) GOTO 1004
I3 = IAC
177 IF(A(I3).NE.0) THEN
    IF(I3.EQ.M) THEN
        A(I3) = 0
        GOTO 1004
    ENDIF
    A(I3) = A(I3+1)
    I3 = I3 + 1
    GOTO 177
ENDIF
GOTO 1004
ENDIF
I1 = A(IAC)
I2 = A(IC)
IF( T(IAC) .EQ. ISHIP ) THEN
    A(IAC) = A(IC)
ELSE
    DO 191 ID = IAC+1, M, 1
        IF(A(ID) .EQ. 0) THEN
            A(ID) = A(IC)
            GOTO 1002
        ENDIF
191 CONTINUE
ENDIF
1002 T(I2) = ISHIP
T(I1) = T(I1) - ISHIP
1004 IA = I
IAC = IC
I = P(I)
GOTO 151
ENDIF
ISS = BJ(DEFI(DF)) - DEM(DEFI(DF))
IF (ISS.LT.0) GOTO 88
DF = DF + 1
GOTO 80
170 IF (.NOT.SWAP) THEN
    WRITE(6,200)' THE ASSIGNMENT PROBLEM IS INFEASIBLE !!!'
    GOTO 270
ENDIF
T1 = CTIME()
T10 = T1 - T0
WRITE(4,190)'*** OPTIMAL SOLUTION REACHED ***'
WRITE(4,230)
WRITE(4,240)
WRITE(4,250)
TOTC = 0
DO 180 I=1,N
    IC = 1
1001 IF (A(IC).EQ.0) GOTO 180
WRITE(4,200)I,A(IC),AL(A(IC)),T(A(IC))
TOTC = TOTC + AL(A(IC)) * T(A(IC))
IC = IC + 1
```



```

GOTO 1001
180 CONTINUE
WRITE(4,210)
WRITE(6,260)TOTC
WRITE(6,*)'CPU TIME (SECONDS) :',T10
WRITE(4,280)
190 FORMAT(/17X,A,/29X,'FOR THE',/22X,'TRANSPORTATION PROBLEM ')
200 FORMAT(15X,'| ',13,' | ',13,' | ',15,' | ',15,' | ')
210 FORMAT(15X,'-----')
230 FORMAT(/15X,'-----')
240 FORMAT(15X,'| FROM | TO | COST | SHIPMENT |')
250 FORMAT(15X,'|-----|')
260 FORMAT(/15X,' TOTAL TRANSPORTATION COST = ',16)
280 FORMAT(/21X,' *** END OF OUTPUT ***')
270 STOP
END

```

Appendix H

C
 C THIS PROGRAM CREATED RANDOM NETWORKS FOR ASSIGNMENT, SEM-ASSIGNMENT
 C AND TRANSPORTATION NETWORKS. THE LOGIC OF THIS PROGRAM IS BASED ON
 C NETGEN CODE BY KLIGMAN.

C
 C NOTE : TO RUN THIS PROGRAM FOR OTHER CODES THE FORMAT FOR THE
 C PRINT STATEMENTS NEED TO BE CHANGED, BECAUSE EVERY CODE
 C HAS IT'S STRUCTURE FOR THE INPUT DATA FORMAT. THE PRINT
 C STATEMENTS OVER HERE ARE FOR THE SSP ALGORITHM.

C
 C INTEGER SEED, SUP(100), NUA(2000), DES(100, 100), DEM(100), AJ(100)
 C REAL U(150)
 C DATA N, M, NARC, MCOST, TOTSUP/100, 100, 1300, 100, 100000/
 C OPEN(1, FILE='P53 DATA')
 C SEED=13502460
 C DO 5 I=1, 10000
 C SEED = RANUN(SEED)
 5 CONTINUE

C
 C CREATING SUPPLY FOR EACH SOURCE NODE
 C

SUM = 0
 DO 10 I=1, N
 U(I) = RANUN(SEED)
 SUM = SUM + U(I)
 10 CONTINUE
 ISUM = 0
 DO 20 I=1, N-1
 U(I) = U(I) / SUM
 SUP(I) = TOTSUP * U(I)
 IF(SUP(I) .EQ. 0) SUP(I) = 1
 ISUM = ISUM + SUP(I)
 WRITE(1, *) SUP(I)
 20 CONTINUE
 SUP(N) = TOTSUP - ISUM
 WRITE(1, *) SUP(N)

C
 C CREATING DESTINATIONS
 C

SUM = 0
 DO 30 I=1, N
 34 U(I) = RANUN(SEED)
 IF(U(I).GT.0.98) U(I) = 1
 ANARC = REAL(NARC)
 AM = REAL(M)
 NUA(I) = U(I) * (ANARC / AM)
 IF(NUA(I).LT.1) GOTO 34
 SUM = SUM + NUA(I)
 30 CONTINUE
 ISUM = NARC - SUM
 44 IF(ISUM.NE.0) THEN
 45 XU = RANUN(SEED)
 IF(XU.GT.0.98) XU = 1
 JX = XU * N
 IF(JX.EQ.0) GOTO 45

```

IF(NUA(JX).LT.M) THEN
  IA = MIN((M-NUA(JX)), ISUM)
  NUA(JX) = NUA(JX) + IA
  ISUM = ISUM - IA
ENDIF
GOTO 44
ENDIF

```

```

C
C CREATING DESTINATIONS ASSIGNMENTS
C

```

```

DO 42 J=1,M
  DFM(J) = 0
  AJ(J) = 0
42 CONTINUE
DO 50 I=1,N
DO 70 J=1,NUA(I)
77  U(J) = RANUN(SEED)
  IF ( U(J) .GT. 0.99) U(J) = 1
  XD = M * U(J) + 0.05
  DES(I,J) = INT(XD)
  IF(DES(I,J) .EQ. 0) GOTO 77
  DO 71 K=1,J-1
    IF(DES(I,J).EQ.DES(I,K)) GOTO 77
71  CONTINUE
  IJ = DES(I,J)
  AJ(IJ) = AJ(IJ) + 1
70 CONTINUE
50 CONTINUE
DO 51 J=1,M
52  IF( AJ(J) .EQ. 0 ) THEN
    RN = RANUN(SEED)
    IS = N * RN
    DO 12 I=1,NUA(IS)
      IJ = DES(IS,I)
      IF ( AJ(IJ) .GT. 1 ) THEN
        DES(IS,I) = J
        AJ(J) = AJ(J) + 1
        AJ(IJ) = AJ(IJ) - 1
      ENDIF
12  CONTINUE
    GOTO 52
  ENDIF
51 CONTINUE

```

```

C
C CREATING DEMAND FOR EACH DESTINATION
C

```

```

DO 75 I=1,N
  SUM = 0
  DO 80 L=1,NUA(I)
    U(L) = RANUN(SEED)
    SUM = SUM + U(L)
80 CONTINUE
  ISUM = 0
  DO 90 IL=1,NUA(I) - 1
    U(IL) = U(IL) / SUM

```

```
      IJ = DES(I,IL)
      DEM(IJ) = SUP(I) * U(IL) + DEM(IJ)
      ISUM = ISUM + SUP(I) * U(IL)
90  CONTINUE
      IL = NUA(I)
      IJ = DES(I,IL)
      DEM(IJ) = DEM(IJ) + ( SUP(I) - ISUM )
75  CONTINUE
      DO 31 I=1,M
      WRITE(1,*)DEM(I)
31  CONTINUE
C
C  CREATING COST FOR EACH ARC
C
      DO 15 I=1,N
      DO 16 J=1,NUA(I)
      IJ = DES(I,J)
301  R = RANUN(SEED)
      R = R * MCOST
      IR = INT(R)
      IF(IR.EQ.0) GOTO 301
      WRITE(1,*)I, IJ, IR
16  CONTINUE
15  CONTINUE
      STOP
      END
C
C  FUNCTION TO GENERATE REAL RANDOM NUMBERS BETWEEN 0. AND 1.0
C  INPUT :
C      IX: ODD INTEGER ( NO MORE THAN NINE DIGITS )
C
      FUNCTION RANUN(IX)
      IX=MOD(25173*IX+13849,65536)
      RANUN=REAL(IX)/65536
      RETURN
      END
```

Vita

Majed Saleh Al-Ghassab was born in Al-Khobar, Saudi Arabia, on May 28, 1968. After graduating from Ibn-Sina Secondary School, he attended King Fahd University of Petroleum and Minerals at Dhahran. He received the degree of Bachelor of Science with a major in Systems Engineering in August 1990. In September 1990, he was employed in the office of the Registrar at the same university as a Systems Analyst. Meanwhile, he was doing masters as a part-time student. In November 1992, he joined Saudi Petrochemical Company (SADAF) as a Systems Analyst.