



5-1999

A spatial decision support system for designing solid waste collection routes in rural counties

Xiaohong Xin

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

Recommended Citation

Xin, Xiaohong, "A spatial decision support system for designing solid waste collection routes in rural counties. " Master's Thesis, University of Tennessee, 1999.
https://trace.tennessee.edu/utk_gradthes/10056

This Thesis is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Xiaohong Xin entitled "A spatial decision support system for designing solid waste collection routes in rural counties." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Geography.

Bruce Ralston, Major Professor

We have read this thesis and recommend its acceptance:

Thomas Bell, John Rehder

Accepted for the Council:

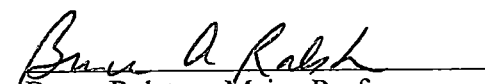
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

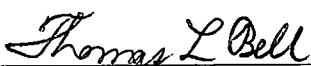
(Original signatures are on file with official student records.)

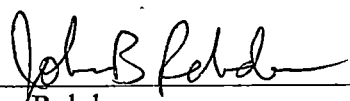
To the Graduate Council:

I am submitting herewith a thesis written by Xiaohong Xin entitled "A Spatial Decision Support System for Designing Solid Waste Collection Routes in Rural Counties." I have examined the final copy of the thesis for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree Master of Science, with a major in Geography.



Bruce Ralston, Major Professor

We have read this thesis
and recommend its acceptance:


Thomas Bell


John Rehder

Accepted for the Council:


Associate Vice Chancellor and
Dean of The Graduate School

**A SPATIAL DECISION SUPPORT SYSTEM FOR
DESIGNING SOLID WASTE COLLECTON ROUTES
IN RURAL COUNTIES**

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Xiaohong Xin
May, 1999

ACKNOWLEDGEMENTS

I can not remember how many times, when I walked breathlessly to the Geography Building on the hill, I blamed for these endless stairs. At the moment when I finally finish this thesis, I realize that I have owned much debt to that path leading to the top of the hill. Many colleagues, family members, and friends have been helping me and walking along with me no matter what I experience.

I would like to express my deepest gratitude to Dr. Bruce Ralston as a wonderful advisor and caring person. His vision and wisdom can lead me to go very far away whenever I stand at a dead end, which creates pleasurable moments associated with learning that will always be a sweet memories. His humor and appreciation always reminds me of finding solutions quickly if I just turn around to look at the other side of things. I would like to thank the other members of my Thesis Committee – Dr. Thomas Bell and Dr. John Rehder, for acceptance, guidance, and sharing their knowledge. I must also offer my thanks to Dr. Cheng Liu and his family. They have given me so much help and encouragement during these years. Of course, my appreciation is for Chris Garkovich, my boss in County Technology Assistance Service of UTK, for her confidence in me and her financial support, which makes the whole thesis research possible.

My family members have been my example and strong support. My parents instilled hardworking spirit to me when I was a child. My sisters and sister-in-laws keeps on telling me to go wherever I can go and do not look back, but I know they keep eyes on me forever and they will sacrifice everything for me.

Most importantly, my thanks go to my friends in Knoxville Chinese Christian Campus Fellowship, for their prayers to help me be faithful, loving, and humble and for the promise that I am not alone on my way to the hardest and the highest.

ABSTRACT

Because of the increasing need to develop optimized routes for solid waste collection in rural counties, there has been a lot of research on exploring arc routing problems and their efficient solutions. The complex nature of solving arc routing problems lends itself to a GIS-based spatial decision support system. Such a system could combine user knowledge of a problem with heuristic algorithms to obtain arc routing solutions. This thesis presents SWRoute, which is such a system.

Developing SWRoute requires three major steps: building a suitable geographic data base, determining and implementing the necessary heuristic algorithmic techniques, and setting up a GIS framework that allows users to manipulate the data inputs and the algorithm outputs. Taken together, these three components form a spatial decision support system for designing solid waste collection routes.

The area of study is Hamblen County in East Tennessee. A database of roads in the county was developed using TIGER and other map sources. The demands for solid waste collection were obtained from the Hamblen County solid waste region.

SWRoute also uses two modeling algorithms. The first is a heuristic algorithm for generating solutions to the rural arc routing problem. The second algorithm is used to develop lower bounds on candidate solutions. The lower bounds help determine the quality of the heuristic solutions.

The SWRoute interface has tools which allow the user to partition a base road network into subnetworks and create a seed node set. The GIS interface is also useful for

generating and reporting routes. With the aid of these tools, the user can study and manipulate solutions generated from the algorithms.

The results derived from the Hamblen County example indicate how a GIS-based spatial decision support system can help solve complex problems facing rural U.S. counties.

TABLE OF CONTENTS

CHAPTER	PAGE
1. STATEMENT OF THE PROBLEM.....	1
1.1 Introduction.....	1
1.2 Arc Routing Problems.....	2
1.3 SWRoute: a GIS-based Spatial Decision Support System for Designing Solid Waste Collection Routes.....	3
1.4 Scope of the Research.....	4
1.5 Inspiration for the Research.....	5
1.6 Concluding Remarks.....	7
2. METHODOLOGIES OF DEVELOPING SWROUTE.....	9
2.1 Introduction.....	9
2.2 Spatial Decision Support System (SDSS).....	9
2.3 CTAS's Requirements for Developing SWRoute.....	11
2.3.1 System Requirements.....	12
2.3.2 Database Requirements.....	12
2.3.3 Modeling Requirements.....	14
2.3.5 Interface Requirements.....	15
2.4 Characteristics and Assumptions of the Hamblen County SWRoute.....	15
2.4.1 Definition of the Hamblen County SWRoute Region.....	15
2.4.2 SWRoute Solid Waste Generation Characteristics.....	16
2.4.3 Crew Working Time Assumptions.....	19
2.4.4 Collection and Vehicle Assumptions.....	19
2.4.5 Collection and Transportation Regional Facilities.....	20
2.5 Concluding Remarks.....	20
3. MODELING SWROUTE: A CAPACITATED ARC ROUTING PROBLEMS.....	22
3.1 Introduction.....	22
3.2 Capacitated Arc Routing Problems (CARP).....	25
3.2.1 Augment-merge Algorithm Notation and Assumptions.....	27
3.2.2 Algorithm Statement and Implementation.....	27
3.2.3 An Example of Augment-merge Algorithm.....	32
3.3 CARP Lower Bound Computation.....	40
3.3.1 Algorithm Statement and Implementation.....	40
3.3.2 An Example of Lower Bound Computation for Figure 3.2 Network.....	42
3.4 Data Structures and Their Implementation for Solving CARP and NMCMP.....	49
3.4.1 Doubly Linked Lists and CARP's Augment-merge Algorithm.....	49
3.4.2 Disjoint Sets and NMCMP's Primal-dual Blossom Algorithm.....	50
3.6 Concluding Remarks.....	55

4. SWROUTE INTERFACE DESIGN AND SYSTEM INTEGRATION.....	56
4.1 Introduction.....	56
4.2 SWRoute Base Road Network.....	58
4.2.1 Generating a SWRoute Base Road Network for Hamblen County..	58
4.2.2 SWRoute Base Road Network Data Attributes.....	63
4.3 SWRoute Subnetworks.....	63
4.3.1 Generating the Hamblen County SWRoute Subnetworks.....	63
4.3.2 Data Attributes for a SWRoute Target Network and Subnetworks.....	78
4.4 SWRoute Seed Node Set.....	80
4.4.1 Creating a Seed Node Set for the Hamblen County SWRoute.....	80
4.4.2 Data Attributes for a SWRoute Seed Node Set.....	83
4.5 SWRoute routes and Routing Reports.....	83
4.5.1 Generating SWRoute Routes and Reporting Routes.....	83
4.5.2 Data Attributes for SWRoute Routes.....	90
4.6 Concluding Remarks.....	91
5. Discussion and Recommendations.....	92
5.1 Summary of Research Results.....	92
5.2 Recommendations for Future Research.....	95
Bibliography.....	99
Appendices.....	105
Appendix A CARP Mathematical Formulation.....	106
Appendix B Undirected Matching Network and Nonbipartite Minimum Cost Perfect Matching Problem (NMCPMP).....	111
Appendix C AVENUE Pseudocodes for SWRoute Interface Design.....	138
Appendix D C Source Codes for Solving Capacitated Arc Routing Problems..	162
VITA.....	198

LIST OF FIGURES

FIGURE	PAGE
Figure 2.1 The Hamblen County SWRoute Region and its Transportation Network.....	17
Figure 3.1 A Flow Diagram of the Augment-Merge Algorithm.....	31
Figure 3.2 A CARP Network.....	32
Figure 3.3 A Flow Diagram of Computing CARP Lower Bound.....	43
Figure 3.4 A Network when $P = 0$	44
Figure 3.5 A Network when $P = 2$ in Case 1.....	46
Figure 3.6 A Network when $P = 2$ in Case 2.....	47
Figure 3.7 A Network when $P = 4$	48
Figure 3.8 A Schematic Description of a Doubly Linked List for SWRoute.....	51
Figure 3.9 A Schematic Description of Doubly Linked Lists for all Routes in a Network	52
Figure 3.10 A Schematic Description of Disjoint Sets and their Related Structure Components.....	54
Figure 4.1 SWRoute Interface Overview.....	57
Figure 4.2 A Descriptive Flow Diagram for SWRoute Objects (Black Letters) and their Interface Representation.....	57
Figure 4.3 Flow Diagram of Generating a SWRoute Base Road Network.....	61
Figure 4.4 A SWRoute Base Road Network Interface.....	62
Figure 4.5 A Menu for Generating SWRoute Base Road Networks.....	62
Figure 4.6 A Menu for Generating SWRoute Subnetworks.....	67
Figure 4.7 Flow Diagram of Generating SWRoute Subnetworks.....	68

FIGURE	PAGE
Figure 4.8 A Driver Partition Interface after Set up.....	70
Figure 4.9 A Driver Partition Report Written in a MSOffice Document.....	71
Figure 4.10 A Driver Partition Interface with Links being Partitioned.....	72
Figure 4.11 An Interface for Driver-Weekday Partititon.....	74
Figure 4.12 A Driver-Weekday Partition Report Generated in a Notepad.....	75
Figure 4.13 A Scaled Map for Driver 272 Subnetworks.....	76
Figure 4.14 A Driver-Weekday Partition Interface with Subnetworks.....	77
Figure 4.15 A Menu for Building SWRoute Seed Node Sets.....	81
Figure 4.16 Menus and Tools for Generating SWRoute Seed Node Sets.....	81
Figure 4.17 A Dialog Box Used for Adding a Garage.....	82
Figure 4.18 A Dialog Box Used for Editing a Depot.....	82
Figure 4.19 Menus and a Tool for Generating and Reporting SWRoute.....	84
Figure 4.20 A Route Configuration Dialog Box.....	84
Figure 4.21 A Message Box for Checking Seed Nodes and their Associated Routes.....	84
Figure 4.22 A Dialog Box for Showing/Hiding Routes.....	86
Figure 4.23 Routes for Driver 272 on Monday, Tuesday, and Wednesday.....	87
Figure 4.24 A Notepad Route Report for Driver 272 on Wednesday.....	88
Figure 4.25 A Graphic Route Report for Driver 272 on Wednesday.....	89
Figure B.1 A Matching Network with Thick Matching Edges.....	113
Figure B.2 A Simple Blossom.....	114
Figure B.3 A Flow Diagram of Primal-Dual Blossom Algorithm.....	126
Figure B.4 A Derived Matching Network without Blossoms.....	127

FIGURE	PAGE
Figure B.5 A Matching Network with Blossoms.....	131
Figure B.6 A Matching Network with a Shrunk Blossom and A Pseudonode.....	135

LIST OF TABLES

TABLE	PAGE
Table 4.1 Data Attributes in a Base Road Network.....	64

CHAPTER 1

Statement of the Problem

1.1 Introduction

Population growth in rural areas, along with the increased generation of solid-waste by Americans, has caused waste management to become a major expense for local governments. The collection and transportation of solid-waste from sources to landfills is a costly budget item for most rural counties. Therefore, efficient routing of solid-waste collection is an important part of a solid-waste management system. The Office of Solid-Waste Management Program in the U.S. Environmental Protection Agency is increasingly requiring counties to develop planning models to assess the efficiency of solid-waste collection crews and evaluate their routes. Several counties, primarily those with dense populations, have already initiated such evaluation plans for routing, but they are not scientifically rigorous and their methods are not transferable between different counties (Institute for Solid-waste of the American Public Works Association, 1975).

The theoretical structure of optimizing solid-waste collection routes is called an arc routing problem. Although many algorithms have been developed for arc routing problems, they do not in general provide strictly optimal answers (Keenan, 1996). This fact reflects the difficulty of the problem involved and implies that generated routes must be frequently improved using skilled intervention by experienced decision makers. That is, routing should be part of a decision support system. This thesis presents a Geographic Information System (GIS) based on spatial decision support system (SDSS) for use in counties planning a solid-waste routing system.

1.2 Arc Routing Problems

The routing of vehicles is one of the most developed areas of Operations Research. Bodin and Golden (1981) developed a taxonomy for various vehicle routing problems composed of two major classes: node routing problems and arc routing problems. One of the important differences between these classes is that node routing problems are concerned with demands located at nodes and arc routing problems are concerned with demands located along arcs. Node routing problems include shortest path problems and traveling salesman problems. In reality, many problems belong to arc routing problems. Besides solid-waste collection, these also include many problems as postal delivery, meter reading, street sweeping, and snow clearance. Although arc routing problems are of practical interest, there has been less research done in this area than in node routing problems. Consequently, algorithmic techniques in this area are not considered to be as well-developed as node routing problems (Eiselt, Gendreau and Laporte 1995a, 1995b). This reflects the complexity of these problems and the difficulty of taking into consideration many practical issues that can not always be captured in a purely mathematical model. As a result, while many operations research algorithms have been developed for arc routing problems, those do not in general provide strictly optimal solutions (Keenan, 1996). There are several aspects of arc routing problems which need to be considered.

First, arc based problems by definition require attention to the details of road networks. Supposedly, arc routing problems often take place in a small area, usually an urban one. In such an area, a variety of characteristics of the road network become relevant, such as one-way streets and left or right turn prohibitions. An effective decision

support system for such a problem must accommodate all relevant information. A mathematical model is built on the basis of a number of assumptions and it is very difficult for one model to handle all of the above road information (Keenan, et al. 1996).

Secondly, arc routing problems are static in nature (Keenan, et al. 1996). When a route is designed, it may not need to be changed for a given period of time. In the solid-waste case, the collection routes will only be changed in response to changes in road topology or changes in population distribution in an area. Although there may be day-to-day variation in the addresses visited, the same set of streets must be traversed.

The necessity of accessing large amounts of information and the stability of routes over time reflect the distinctive characteristics of arc routing problems. Arc routing problems can not simply borrow classic modeling methods designed for node routing problems. A GIS-based routing system facilitates the design of a spatial decision support system (SDSS) for finding solutions to arc routing problems.

1.3 SWRoute: a GIS-Based Spatial Decision Support System for Designing Solid-waste Collection Routes

Geographic Information System (GIS) is not just a software system, but a sophisticated science. Its development “tackles the scientific questions that geographic information handling raises and pursues scientific goals using the technology that the system provides” (Worboys, 1995). Accordingly, a decision support system (DSS), combined with a GIS system, not only provides tools to handle information but also addresses theoretical issues about how real world problems are abstracted in order to take advantage of DSS’s analytical ability.

A decision support system with spatial analytical abilities is called spatial decision support system (SDSS). When it is integrated with GIS, a system built with spatial decision making support functions is thought of as a GIS-assisted spatial decision support system. The system is usually defined to consist of a database component, modeling components, and a user interface component. It is usually assumed that the system should allow the use of modeling techniques with a facility for user intervention.

A strong link between GIS and SDSS requires an ongoing process that could take many years to complete (Sieg, 1988). There are three benefits associated with this linkage. The first benefit is efficiency, i.e., the time and labor saved due to the automation of processes. The second is effectiveness, which can be defined as the enhancements made to decision making based on the availability of information to the user. The third aspect concerns unexpected intangible benefits, which are related to the improvement of other associated systems. Designing solid-waste collection routes can be regarded as a GIS-assisted SDSS application (Peenan, 1996), which should achieve these benefits.

1.4 Scope of the Research

This thesis deals with a methodology of integrating GIS and SDSS. Concepts and algorithms of arc routing problems are explored.

A major question addressed in this thesis is:

Can a prototype GIS-assisted system be built with decision support functions, which can be used to develop, apply, and evaluate a wide range of solid-waste collection route systems in U.S. rural counties?

This main question leads to several other questions to be answered in this thesis research:

Question 1: What are the advantages and the disadvantages of a methodology of linking GIS and SDSS for designing solid-waste collection routes?

Question 2: Can we classify the algorithms currently used for solving arc routing problems into useful sets of approaches and processes?

Question 3: What kinds of heuristic algorithms present a reasonable solution to the solid-waste collection routes?

Question 4: Can we build general-purpose tools to match those algorithms?

The remainder of the thesis has the following sections. In the first section, a discussion of the problem facing Hamblen County, Tennessee is presented. The county serves as a test case for SWRoute, so a thorough understanding of its routing system is required. Next, the thesis specifies the requirements and assumptions for building a prototype SDSS. The second section concentrates on the theoretical models and algorithms of SWRoute. Data structures and their contributions to improve the algorithms are also discussed. The third section presents a detailed description of the interface design for SWRoute.

1.5 Inspiration for the Research

The County Technology Assistance Service (CTAS) at the University of Tennessee intends to build a well-equipped GIS in its three regional offices (Knoxville, Nashville, and Jackson). Designing a GIS-based SDSS for solid-waste collection routes

in several counties of Tennessee is CTAS's first project. This thesis is part of a prototype study, which should be developed to be transferable between different counties.

Once largely ignored, arc routing problems are increasingly being recognized as important components of vehicle routing problems.

Many practical problems can be modeled as arc routing problems, so there is renewed interest in designing add-in modules or specialized software to obtain arc routing solutions. As early as 1993, WastePlan, a microcomputer-based solid-waste planning model originally developed by the Tellus Institute in Boston for the Office of Technology Assessment, included one interactive program for solid-waste collection systems. However, its functions are so limited that only specific factors such as physical and financial characteristics of different trucks and containers, crew size, average miles from the route to processing facilities are defined. In that system, no real routings along streets are modeled and mentioning "the routes" just refers to a situation that waste streams are moved from sources to any number of potential processing facilities. As a matter of fact, the problem addressed in WastePlan is a node routing problem.

In 1989, an authentic vehicle routing and scheduling software product, RouteSmart, was released under the auspices of RouteSmart Technologies and four University of Maryland professors, Arjang Assad, Michael Ball, Lawrence Bodin and Bruce Golden. Widely used by city and county public works departments, applications of RouteSmart include sanitation collection routing, utility meter reading, newspaper distribution and other local pickup, and delivery services. RouteSmart has two modules, a workload estimation and route partitioning module and a travel path generation module, which is used for neighborhood vehicle routing. Depending on the nature of the routing

scenario, either one or both modules may be needed. The market price of one license of RouteSmart software is \$77,000 plus costs for computers and mapping data. Thus, cost prevents solid-waste management departments in small-to-medium counties from purchasing a whole system for a simple routing run. Designing an arc routing system within a reasonable cost range is my endeavor and a goal for CTAS solid-waste routing projects.

In University College at Dublin in Ireland, P. Keenan, a Management Information System (MIS) professor has studied arc routing for rural Irish networks and developed a decision support system for arc routing. Some meaningful results were achieved by his efforts of integrating GIS and SDSS with arc routing. We contacted each other by email one year ago. He kindly sent two papers to me. Those papers influenced my own research. The idea of building a GIS-assisted SDSS solid-waste routing system has benefited from his work and encouragement.

1.6 Concluding Remarks

This chapter is a general introduction for SWRoute. It began with the clarification of the difference between node routing problems and arc routing problems. As an arc routing problem, SWRoute is considered as a system to be built without mature algorithmic support, at least compared to node routing problems. A GIS-based arc routing system with spatial decision support functions, therefore, is proposed for generating solutions to arc routing problems. Definitions of a decision support system (DSS) and a spatial decision support system (SDSS) are then briefly introduced to confine the thesis research scope and associated questions to be answered at the end of the thesis research.

The next chapter will discuss the methodology of developing a system with GIS-assisted SDSS functions for designing solid-waste collection routes.

CHAPTER 2

Methodologies of Developing SWRoute

2.1 Introduction

This chapter presents some principles for GIS-based Solid-waste Routing SDSS (SWRoute) design methodologies. Since SDSS is introduced as a framework of developing SWRoute, the chapter first identifies the characteristics of SDSS, the component of SDSS, various levels of SDSS technology and the different organizational roles associated with SDSS. Then, the process of designing SWRoute in Hamblen County Tennessee is presented in order to address requirements and characteristics for a solid-waste collection system.

2.2 Spatial Decision Support System (SDSS)

Complex spatial problems often have to meet multiple criteria with more than one objective or goal. Classical methods assume that the problem is sufficiently precise such that the goal and objectives can be defined. However, many problems are ill-structured in the sense that the goals and objectives are not completely defined. Decision Support Systems (DSS) provide formal procedure for the application of well-structured models to poorly structured decision problems (Batty, 1990). Particularly, DSS is an interactive system that provides users with easy access to decision models and data in order to support poorly structured decision-making tasks (Sprague, 1989). The range of decision problems to which these models might be applied is considerable. Difficulties arise over

how to identify principles for good model design and application. Also, these principles should be consistent with methods for analyzing and visualizing geographic data.

GIS imposes quite severe constraints on the way that geographic data can be represented, analyzed, and displayed (Batty, 1990). Although it might be possible to loosely-couple GIS with appropriate DSS based on some spatial model applications, GISs fall short of the goals of DSS for a number of reasons (Densham, 1990):

1. analytical modeling capabilities often are not part of a GIS;
2. many GIS databases have been designed solely for cartographic display of results, but DSS goals require flexibility in the way information is communicated to the user;
3. the set of variables or layers in the database may be insufficient for complex modeling;
4. data may be at insufficient scale or resolution;
5. GIS designs may not be flexible enough to accommodate variations in either the context or the process of spatial decision-making.

Spatial DSS (SDSS) are often designed virtually from scratch in order to provide a flexible problem-solving environment for complex spatial problems.

Several authors (Craig and Moyer, 1991; Goodchild and Densham, 1990; Moon, 1992; NCGIA, 1992; Ralston, 1993) have extensively covered similar descriptions of characteristics for SDSS. Their works define SDSS as an *interactive* computer-based information system, which *helps* decision-makers utilize *geographic data* and *spatial models* to solve *unstructured* problems. The unique contribution of SDSS results from the above key words. The definition is quite demanding, and few existing systems completely satisfy its requirements. The definition of SDSS has recently been extended to

include any system with “intuitive validity” that makes some contributions to spatial decision making. Accordingly, a broader definition of SDSS, described by Donald Cooke(1992), is a “canned software that is intuitively obvious to use, solves problems efficiently and delivers immediate results”. In other words, in Cooke’s view, SDSS becomes off-the-shelf software for carefully selected functions with bug-free point-and-shoot capabilities on very specific spatially oriented needs. In this way, SDSS doesn’t require in-depth commands to operate, yet allows users to negotiate very sophisticated geographic analysis.

A SDSS about solid-waste collection routes is often complex and difficult to design because many factors must be considered and a wide range of routing options is available. Solid-waste collection routes in different areas vary greatly depending on the waste type collected, the characteristics of the area, and the preference of its residents. Often, different solid-waste collection equipment, methods, or service providers are required in the same area to serve different customers or to collect different materials from the same customers. To simplify the system design and modification, the following sections address requirements and assumptions for developing SWRoute to best meet a specific area’s need.

2.3 CTAS’ Requirements for Developing SWRoute

From the start of this research, CTAS was concerned about what a SWRoute system should look like. However, no documentation exists which described the specific requirements for such a system. Since this was the first use of a GIS system in CTAS, the concepts of GIS and its application to SWRoute did challenge people in CTAS to

think about their GIS requirements. Therefore, the requirements addressed in the following section are not only for the SWRoute project but also for building a GIS system served for other county-wide projects in a Solid-waste Department. In general, they are categorized with respect to system requirements, database requirements, modeling requirements, and interface requirements.

2.3.1 System Requirements

SWRoute was designed to operate on a stand-alone personal computer running Microsoft Windows 95. The Windows 95 operating system should allow integration of ESRI GIS software and other window-based software, for example Borland C++ compiler, Borland Delphi, and Microsoft Word. UNIX ARC/INFO in a Xterminal UNIX window is needed to edit network coverages. Although the faster CPU is desired for calculation speed, the system can operate using a standard Intel Pentium microcomputer running at 200 Mhz and 32 Mb of RAM. Scanners, digitizers, and color printers are required for preparing data, and printing route reports for drivers.

2.3.2 Database Requirements

Since a database is the foundation of any GIS-based SDSS, CTAS developed a detailed description about the information to be included in its GIS database. The database not only serves the SWRoute project but also other projects. In general, CTAS has following requirements with respect to database design:

1. A long-term methodology is formulated to build and improve the SWRoute GIS database;

2. Constructing the database should foster cooperation among county agency GIS activities;
3. Database building should eliminate duplication of effort in digitizing information among county agencies;
4. SWRoute project enhances data sharing between county agencies by identifying and adopting standards for use in digitizing information.

Technically, the SWRoute database consists of several independent layers of data, all linked to the same coordinate system. Data should be maintained at the lowest level of disaggregation and then readily aggregated as the need arises.

A digital database with information on a road network should be obtained to determine street configuration. A digital database of the road network should contain two types of information, coordinate data and attribute data. The former indicates the geographic location of nodes or street intersections, and the latter describes characteristics of links and the system. The set of attribute data in the database can include a multitude of variables needed to support the intended applications. For geocoding, address ranges must be stored as attributes. The length of the link does not have to be explicitly stored in the database because it can be measured by the GIS.

Data concerning solid-waste generator types and volumes of waste generated should be gathered so that area collection needs can be determined. Estimates of solid-waste generation can usually be developed through a combination of historical data, data from similar areas, and typical published values.

2.3.3 Modeling Requirements

There are three levels of integration of model and GIS-based SDSS. The lowest level of integration, called loose coupling, is characterized by the use of conversion programs and procedures, and data exchanges between models and GIS-based SDSS, using ASCII files in most cases. The medium level of integration is characterized by an automated and transparent procedure for exchanging data, mainly through the use of a common database, and allowing the model to address this database directly. In the high level of integration, the distinction between model and GIS-based SDSS becomes blurred. Powerful toolboxes are used for storing, retrieving and analyzing geographic data and an experienced user can build solutions from these tools to solve many spatial problems.

Modeling arc routing problem in this thesis uses the lowest level of integration of models and GIS. This loose coupling uses separate models and GIS, connecting their data and exchange files. This approach will implement the following procedures:

1. inputting of geographic distributed data through GIS;
2. exporting of GIS-data and converting them into the variables and parameters used into the model;
3. running the routing model;
4. importing the results of the arc routing models;
5. developing interactive analysis of the model-results and creating final maps and reports.

2.3.5 Interface Requirements

The SWRoute graphic user interface was designed using an object oriented principles. SWRoute objects refer to entities with attributes and functions that are applied and implemented for designing solid-waste collection routes. The highest level of SWRoute objects is composed of nodes and links of a base road network. The lowest level of SWRoute objects is a solid-waste collection route generated for SWRoute. Objects at the lower level inherit properties and methods from those at the higher level. Tool boxes are developed for accessing and manipulating these objects, meaning that the ultimate users may not be GIS professionals.

2.4 Characteristics and Assumptions of the Hamblen County SWRoute

Designing optimized solid-waste collection routes is often complex because it must be addressed within the environment of a solid-waste management system in which many system factors must be considered. In this section, the salient aspects of solid-waste management in the study area, Hamblen County, are discussed.

2.4.1 Definition of the Hamblen County SWRoute Region

Hamblen County is located in the eastern part of the East Tennessee Development District. Its land area is 161 square miles, making it the smallest sized county in the East Tennessee Development District. Population density averages 313.5 persons per square mile, making Hamblen County the second most densely populated county in the Development District, ranking behind only Knox County (East Tennessee Development District, 1995). There are three census county divisions and one incorporated area

(Morristown) in Hamblen County. They are Alpha Division, which contains a part of the City of Morristown; the Morristown Division, which contains most of the City of Morristown; and the Whitesburg Division.

The Hamblen County SWRoute Region consists of all of the rural areas of Hamblen County beyond the boundary of the City of Morristown. The region is bounded on the north by Cherokee Lake and on the southeast by the Nolichucky River and Douglas Lake. The dominant land use within the region is residential, ranging from low density rural to medium and higher density in the semi-urban areas. The majority region is developed with paved roads and public water.

The SWRoute Region is not contained within a Metropolitan Statistics Area (M.S.A.), but it is contiguous to the Knoxville M.S.A. The region contains 9.92 miles of interstate highway and 36.96 miles of U.S. primary highway. It also contains 42.33 miles of U.S. secondary and 4322.9 miles of state, county, and other highway and roads ("Hamblen County Solid-waste Regional Plan", 1995). Figure 2.1 illustrates the major road systems, waterways and political boundaries.

2.4.2 SWRoute Solid-waste Generation Characteristics

1. The amount of solid-waste generated from residential sources in Hamblen County is influenced by the population distribution and economic activities in the county. According to the U.S. Census Bureau, the 1993 population of the Hamblen County was 29,095. The region contained 19,429 households and they resided in predominantly single family detached owner occupied units. Mobile homes and multi-family dwelling units share the remainder in almost equal proportions. Multi-

Hamblen County

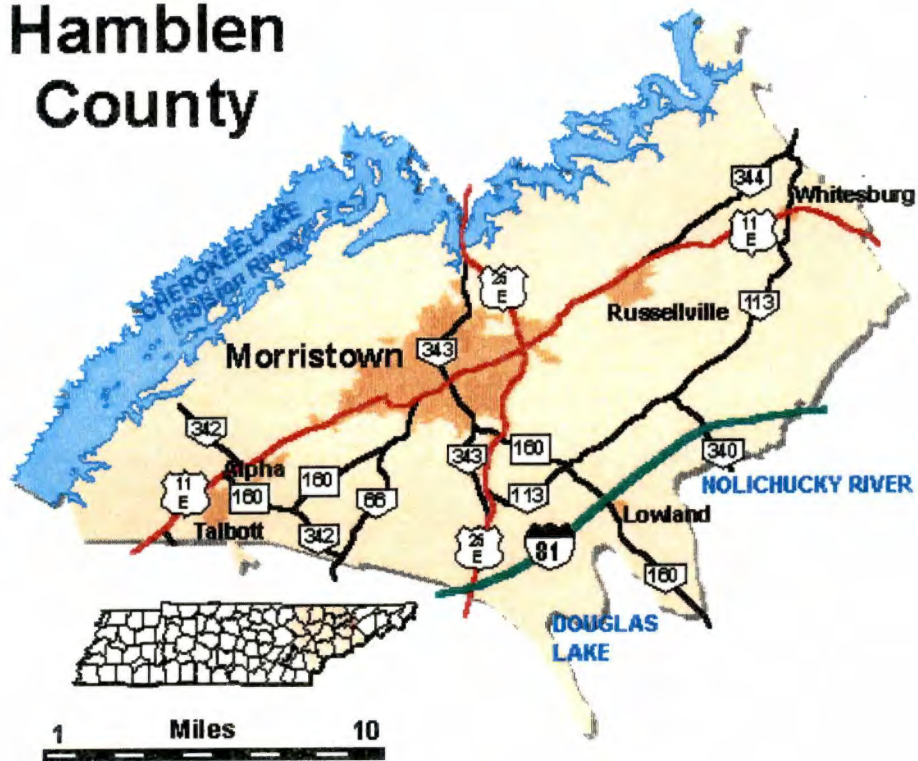


Figure 2.1 The Hamblen County SW Route Region and its Transportation Network

family dwelling units are predominantly renter occupied and mobile homes are predominantly owner occupied.

Several economic factors will affect population growth and solid-waste generation in the next ten years ("Hamblen County Solid-waste Regional Plan", 1995). The development of a third industrial park during the next ten years combined with the expansion of the two existing industrial parks should result in approximately 3,000 new jobs. These new jobs will create expansion in both the commercial and service industries. All of these new jobs will increase the need for expansion of new residential development. In addition, industrial, commercial and service expansion will increase the production of solid-waste.

2. The next ten years will see the extension of public sewer into approximately 25% of the county area which is not presently served. This will result in an increase in population and density in these areas. This could also allow a population expansion of approximately 10,000 persons over the 10-year time span.
3. The completion of the new southern by-pass, which connects the western MAID Industrial Park with Davy Crockett Parkway and the East Tennessee Valley Industrial District will make commuting between the eastern and western parts of the county much quicker. This connector will open up the southern section of the county for residential development

It is estimated in the "Hamblen County Solid-waste Regional Plan" (1995) that, based on the above factors, the population within the next ten years should increase 23.2%. This increase will result in a corresponding increase of the solid-waste generated in the region.

If only residential solid-waste is considered, it is assumed that the solid-waste generating rate for one household per day was about 22.1 pounds in 1991 ("Hamblen County Solid-waste Needs Assessment"). It is assumed that the generation rate will not change during the next ten years.

2.4.3 Crew Working Time Assumptions

I now consider the elements of the solid-waste management system. That system is composed of crews, vehicles, facilities, and routes. The total time for which a crew is paid is usually 8 hours. According to federal law, a crew can not exceed 10 hours of driving without a rest period that allows time to sleep. In addition to driving time, the crew can not work more than 4 additional hours in a shift performing other duties. Therefore, it is assumed to assign a crew to finish his work within 14 hours per day. These facts act as constraints in the routing model.

2.4.4 Collection and Vehicle Assumptions

Hamblen County provides curbside pickup weekly to every citizen within the SWRoute region. Rear loading trucks are used as a major form of collection vehicles. The capacity of the trucks is 17,800 pounds.

The existing collection system will be maintained and increased to meet population increases. Hamblen County will evaluate the incorporation of totally automated loading system to reduce the manpower required and increase efficiency. Presently, the entire SWRoute region is served by four truck drivers. According to the

“Solid-waste Plan By Hamblen County” (1995), the number of drivers will not change within next ten years.

2.4.5 Collection and Transportation Regional Facilities

Every day, the drivers start from a garage, collect solid-waste along their routes, transport solid-waste to either a recycling center or the regional bailing/landfill operation center, and then drive back to the garage.

The garage is located at 125 E. Economy Road. The single waste processing facility within SWRoute region is Lakeway Recycling Center, which is located in Roe Junction or Sublett Road. This facility is presently the only provider of both separation and sale of recyclable materials within the region. It is operated by a private company which accepts recycling materials from their own private collection and from Hamblen County collection. The Hamblen County-Morristown Solid-waste System is the only public landfill facility shared in both management and funding by Morristown and Hamblen County. This facility is also located on Sublett Road. The landfill has a projected capacity of approximately 20 years and will be upgraded to meet all existing and new regulations.

2.5 Concluding Remarks

This chapter presented the principles used in designing SWRoute, a GIS-based SDSS. The design principles focused on how the system should allow users to negotiate very sophisticated geographic analysis, not on the in-depth models and their details. According to some special requirements for a solid-waste collection system proposed by

CTAS, the SDSS for SWRoute should be a PC-based software module, which is embedded in an ArcView interface. Its database should be constructed to be durable and efficient and meet different needs within CTAS. SWRoute adopts a low level of integration of models and GIS, and data exchanges are in the form of ASCII files. Toolboxes are designed with basic operations for manipulating the modeling objects. The final sections of the chapter focus on the study area: Hamblen County, Tennessee. The next chapter makes an in-depth study about the theoretical framework of SWRoute -- capacitated arc routing problems.

CHAPTER 3

Modeling SWRoute: a Capacitated Arc Routing Problem

3.1 Introduction

Uncapacitated routing problems can be classified as node routing problems, arc routing problems, or general routing problems (Bodin, 1975). The traveling salesman problem (TSP) is a node routing problem which solves the problem of visiting all nodes in a network and returning to the starting point. The problem of covering all arcs in a network while minimizing the total distance is the Chinese postman problem (CPP), which belongs to the class of arc routing problems. The general routing problem on network is a generalization which includes the TSP and CPP as special cases. In each case, we assume that all arcs are undirected and that there is a vehicle of unlimited capacity.

Capacitated variations of routing problems include capacitated node routing problems and capacitated arc routing problems. The capacitated node routing problem, known as the vehicle routing problem, has been the focus of much research attention (Beltrami and Bodin, 1974; Boden and Golden, 1981; Christofides et al., 1981; Magnanti, 1981; Golden et al., 1977; and Russell, 1977).

Capacitated Arc Routing Problem (CARP), first introduced by Golden and Wang (1981), was formulated on a directed network. Belenguer and Benavent (1991) then introduced it on an undirected network. In both cases, given $G(N, A, C)^1$, the CARP is defined as a problem that for all arcs $(i, j) \in A$, which must be satisfied by one of a

fleet of vehicles of capacity W , find a set of cycles, each of which passes through the depot node and satisfies demands at a minimal total distance.

If we assume all arcs with demands $q_{ij} > 0$, CARP reduces to Capacitated Chinese Postman Problem (CCPP). If the total demand of all arcs serviced by any particular vehicle does not exceed its vehicle capacity W , i.e. $W \geq \sum_i \sum_j q_{ij}$, then the Chinese Postman Problem (CPP) is obtained from CARP. With one vehicle of capacity $W \geq \sum_i \sum_j q_{ij}$, and $q_{ij} > 0$ for all arcs $(i, j) \in R \subseteq A$, we have the Rural Postman Problem (RPP) which only requires traversing a subset $R \subseteq A$. Thus, CCPP, CPP, and the RPP are all special cases of the CARP.

SWRoute is generally considered to belong to the capacitated arc routing problem (CARP). One of the first authors who studied the waste collection problem with CARP was Stricker (1970). He developed a computerized arc routing algorithm for the urban waste collection problem. This method was tested on real data from the city of Cambridge, Massachusetts. Beltrami and Bodin (1974) used a similar method for New York City, and Negreiros (1974) studied the solid-waste collection system in Rio de Janeiro, Brazil.

Exact algorithms of the CARP have been investigated by Assad, Pearn, and Golden (1987) and by Busch (1991). Their studies showed that if G is a connected and acyclic² network, the vehicle capacity W is the same for all vehicles and all arcs have the same demand $q_{ij} = 1$, then the CARP can be solved in $O(|N|)^3$ time. Assad, Pearn, and

¹ G is a network, N is the set of all nodes, A is the set of all arcs, C is the matrix of distance impedance.

² The network contains no directed cycle.

³ $|N|$ is the absolute value of the number of nodes N , N is a measure of how large of the problem is. $O(|N|)$ is a positive valued function $g(N)$ of the nonnegative variables N is said to be $O(|N|)$ if there exists a constant $g(|N|) \leq O(|N|)$ for all $|N|$.

Golden further addressed that the CCPP with identical demands and defined on a network with cycles is also solvable in $O(|N|)$ time. They also found that the CCPP defined on a strongly connected⁴ network can be solved in polynomial time if all q_{ij} s are no greater than $W/|N|$ when $|N|$ is odd, and no greater than $W/(|N| - 1)$ when $|N|$ is even.

However, the CARP is very difficult to solve on its α -approximation⁵ version. Golden and Wong (1981) showed that if $C = (c_{ij})$ satisfies the triangle inequality⁶, CCPP is attempted to find a solution with a cost less than 1.5 times the value of the optimal solution, which is termed as 0.5-approximate CCPP. They also proved that the 0.5-approximate CCPP is *NP*-complete⁷. Lenstra and Rinooy Kan (1976) proved that the undirected and directed versions of the RPP are *NP*-hard problems. Since the CCPP is a special case of the CARP, a 0.5-approximation version of CARP must be also *NP*-hard.

Due to the computational complexity of the CARP, a standard algorithmic strategy can be used for developing and testing heuristic algorithms. Christofides (1973) presented an algorithm for the CCPP based on an optimal algorithm for the CPP which obtained near-optimal solution to the CCPP. Referred to as the construct-strike algorithm, it significantly outperforms many other existing algorithms on dense networks with small arc demands. The complexity of this algorithm is $O(mn^3)$ ⁸. A modified construct-strike

⁴ The network contains at least one path from every node i to every node j

⁵ The α -approximation version of a problem is defined as a problem of finding a solution whose cost is at most $(1 + \alpha)$ times that of the optimal solution.

⁶ In a network, the cost matrix satisfies the triangle inequality if and only if $c_{ik} + c_{kj} \leq c_{ij}$ for all i, j, k . In arc routing problems, a minimum cost circuit is to find the optimized routes.

⁷ If any of the problems can be solved in polynomial time on a deterministic machine, then all the problems can be solved in polynomial non-deterministic machine ($P = NP$). If the collective failure of all researchers to find efficient algorithms for all of these problems is viewed as a collective failure to prove $P = NP$, such problems are said to be *NP*-complete. The opposite case is called *NP*-hard. A *NP*-completeness proof is a strong indication that the problem is intractable. A *NP*-hard problem implies a possible existence of a polynomially bounded algorithm.

⁸ m is the number of arcs and n is the number of nodes

algorithm (Pearn, 1989) combined minimal spanning tree and matching procedures with the construct-strike method to improve the construct-strike algorithms. The complexity of this algorithm is $O(mn^4)$. Golden, *et al.* (1983) presented two other algorithms called the path-scanning algorithm and the augment-merge algorithm, and Pearn (1989) presented a variation version of the path-scanning algorithm. The complexity of these three algorithms is $O(n^3)$. Finally, in order to implement a heuristic algorithm to handle sparse networks with large arc demands, Pearn (1991) presented the augment-insert algorithm, which combines the merits of the augment-merge algorithm and parallel-insert algorithm (Chapleau, *et al.* 1984). The complexity of the algorithm is still $O(n^3)$.

Although heuristic algorithms solve the CARP problems approximately, they only provide upper bounds of optimal solutions. In order to assess the deviation of the heuristic solutions from optimality, tight lower bounds need to be computed. Golden and Wong (1981) proposed a lower bounding procedure, which was referred as Matching Lower Bound (MLB). It was based on the associated minimal cost 1-matching problem⁹ for obtaining lower bounds for the CARP. Assad, Pearn, and Golden (1987) presented another bounding procedure called the Node Scanning Lower Bound (NSLB), which essentially involved methods of allocating a set of shortest paths that must be added to the original graph for generating the optimal CARP. Pearn (1988) also introduced a bounding method that was based both on the matching lower bound procedure and the node scanning procedure. Pearn (1989) further suggested that the NSLB is expected to perform well on sparse networks. E. Benavent, *et al.* (1992) proposed new lower bounds computed by using a dynamic programming algorithm. Benavent's lower bound is

⁹ The nonbipartite minimum cost perfect matching problem is discussed in Appendix B

theoretically superior to the MLB and NSLB for not only considering the number of vehicles in a fleet needed to cover the whole network but also the number of vehicles required to cover certain sub-networks so that the bounds may be tightened.

The algorithmic aspects of the thesis are primarily concerned with the heuristic algorithm implementation of CARP and the calibration of its lower bound. Since computing lower bounds of a network is based on a matching network, an algorithm about solving a nonbipartite minimum cost matching problem is designed to find pairs of matching nodes in the matching network and the result is further used for lower bound problems. Appendix A provides mathematical programming formulations for CARP and the notation of an undirected matching network. The following sections will firstly present an algorithmic implementation of CARP. Then, an algorithm about solving the non-bipartite minimum cost matching problem along with a method of building a matching network are addressed. Finally, procedures of obtaining lower bounds of CARP are discussed.

3. 2 Capacitated Arc Routing Problems (CARP)

There are several heuristic algorithms which have been suggested in the literature for solving CARP. This thesis implements the augment-merge algorithm. The algorithm, originally introduced by Golden and Wong (1981), is modified and introduced in the thesis research. The following sections 3.2.1 and 3.2.2 present the formal statement of the algorithm. Section 3.2.3 uses an example problem to clarify the implementation of the algorithm.

3.2.1 Augment-merge Algorithm Notation and Assumptions

Let $G^R = (N(R) \cup \{X_0\}, R) = \cup_{k=1}^K G_k^R$ be the subgraph of $G = (N, A, C)$ induced by a node-set N , a route R composed of arc-set A , and a depot node X_0 . Every route R must start and end at depot node X_0 . $G_k^R = (N(R_k) \cup X_0, R_k), k = 1, \dots, K$ are connected cycles of G^R with depot node X_0 .

Let X_{hm} be a head-merging node of R_k , where there is a node $l \in N(R_k)$ such that there may be a $j \in (N \setminus N(R_k)) \setminus \{X_0\}$ with $(j, l) \in A \setminus R_k$ and $j' \in N(R_k)$ with $(j', l) \in R_k$. Let X_{tm} be a tail-merging node of R_k , where there may be a node $l \in N(R_k)$ such that there is a $j \in (N \setminus N(R_k)) \setminus \{X_0\}$ with $(l, j) \in E \setminus R_k$ and $j' \in N(R_k)$ with $(l, j') \in R_k$. Depot node X_0 should not be counted as a head-merging node or a tail-merging node. For example, in a cycle with 1 7 8 3 5 6 4 2 3 1, the underlined parts are non-zero demand arcs that need to be traversed. Node 1 is a depot node. Node 6 is located between two underlined, which can not be assigned as a merge node. Therefore, node 7, node 8, and node 3 are head-merging nodes and node 2 and node 3 are tail-merging nodes.

3.2.2 Algorithm Statement and Implementation

Augment-merge algorithm can be stated as below. Figure 3.1 shows a flow diagram of the iterative procedure.

Phase I: Augment

Step 1: Check Constraints

For every arc (i, j) , check vehicle capacity constraints. If there are arcs with demands $d_{ij} > W$, it is necessary to split the arcs so that no arc with $d_{ij} > W$.

Step 2: Initialize Cycles

For every arc (i, j) with non-zero demand, initial connected cycles are constructed by finding two shortest paths from each endpoint of the arc (i, j) to depot node X_0 so that each cycle services exactly one non-zero arc.

Step 3: Order Cycles

Generated cycles are ordered from longest to shortest according to the length of these routes. Let the longest route be R_1 and the shortest route be R_K ($K = 1, \dots, k$). If every arc (i, j) on a cycle R_{i+1} with a shorter length can be served on cycle R_i with a longer length, the shorter cycle R_{i+1} is discarded. The step is continued until all pairs of cycles have been considered and compared.

Step 4: Label

Starting from the longest cycle R_1 , check every arc (i, j) on the cycle and label those arcs to be traversed. If the demand for arc (i, j) on cycle R_{i+1} can be serviced by the same arc (i, j) on cycle R_i , the demand for arc (i, j) on cycle R_i is not to be serviced by the cycle R_{i+1} . The demand for arc (i, j) on cycle R_{i+1} becomes zero. Also, the demand on its reversed arc (j, i) on cycle R_{i+1} becomes zero. All arcs with non-zero demand are to be traversed. If cycle R_i reaches vehicle capacity or daily time constraints, stop labeling and begin the next cycle R_{i+1} . This step is continued until all cycles have been considered. Finally, check every cycle R_i ($i = 1, \dots, K$) to discard zero-demand cycles,

where:

S_{ij} is the savings attributable to merging.

l_i and l_j are the length of cycle R_i and R_j .

m_{ij} is the length of the cycle resulting from merging cycle R_i and R_j .

Once all possible pairs of (X_{mi}, X'_{hmj}) and (X_{mj}, X'_{imj}) are considered, the merged cycle R'_i with the largest savings is selected to replace original cycle R_i and R_j . R'_i is assigned as the same index as R_i . If there are more than one merged cycle R'_i with the largest savings, arbitrarily select one cycle as R'_i . But, if the capacity of the merged cycle R'_i exceeds the vehicle capacity, original cycle R_i and R_j are kept and no merged cycle are generated.

Step 7: Iterate j

Set $j = j + 1$, go to step 6 until j is greater than the number of cycles.

Step 8: Iterate i

Make a copy of original cycle R_k ($k = 1, \dots, K$), set $i = i + 1$ and keep $i < j$, repeat step 6 and 7 until i is equal to the number of cycles.

Step 8: Reorder Cycles

Return to step 3. Reorder original cycles R_k ($k = 1, \dots, K$). The original cycles $\{R_1, \dots, R_i, \dots, R_k\}$ become $\{R_i, R_1, \dots, R_k\}$. Repeat step 3 through 8 until the cycles $\{R_i, R_1, \dots, R_k\}$ with the minimum total distance are found.

where $\forall (i, j) \in R_i$, is satisfied by $d_{ij} = 0$. This procedure needs to guarantee that the sum of demand of all cycle does not exceed the vehicle capacity. Once this situation happens, the next cycle is labeled instead of continuing the same cycle.

Phase II: Merge

Step 5: Set up Merge Node Subset

For the cycles with at least one non-zero demand arcs, reset cycle indexes as $R_1, \dots, R_i, R_j, \dots, R_k$ ($i < j$). Starting from cycle R_1 , determine the subset of head-merging nodes $\{X_{hm1}, X_{hm2}, \dots, X_{hmn}\} \subset N(R_1)$ and the subset of tail-merging nodes $\{X_{tm1}, X_{tm2}, \dots, X_{tmn}\} \subset N(R_1)$. Then for cycle R_2 , determine the subset of head-merging nodes $\{X'_{hm1}, X'_{hm2}, \dots, X'_{hmn}\} \subset N(R_2)$ and the subset of tail-merging nodes $\{X'_{tm1}, X'_{tm2}, \dots, X'_{tmn}\} \subset N(R_2)$. The step continues until merge node subsets for all cycles are created.

Step 6: Merge Cycles

Starting with cycle R_i and R_j , compare every tail-merging node $X_{tmp} \in \{X_{tm1}, X_{tm2}, \dots, X_{tmn}\}$ ($p = 1, \dots, n$) for R_i with every head-merging node $X'_{hmq} \in \{X'_{hm1}, X'_{hm2}, \dots, X'_{hmn}\}$ ($q = 1, \dots, n$) for R_j . Then compare every tail-merging node $X_{tmp} \in \{X_{tm1}, X_{tm2}, \dots, X_{tmn}\}$ ($p = 1, \dots, n$) for R_i with every tail-merging node $X'_{tmq} \in \{X'_{tm1}, X'_{tm2}, \dots, X'_{tmn}\}$ ($q = 1, \dots, n$) for R_j . If $X_{tmp} = X'_{hmq}$ or $X_{tmp} = X'_{tmq}$, the savings associated with a pairwise merging of the cycle R_i and R_j is evaluated as the expression:

$$S_{ij} = l_i + l_j - m_{ij}$$

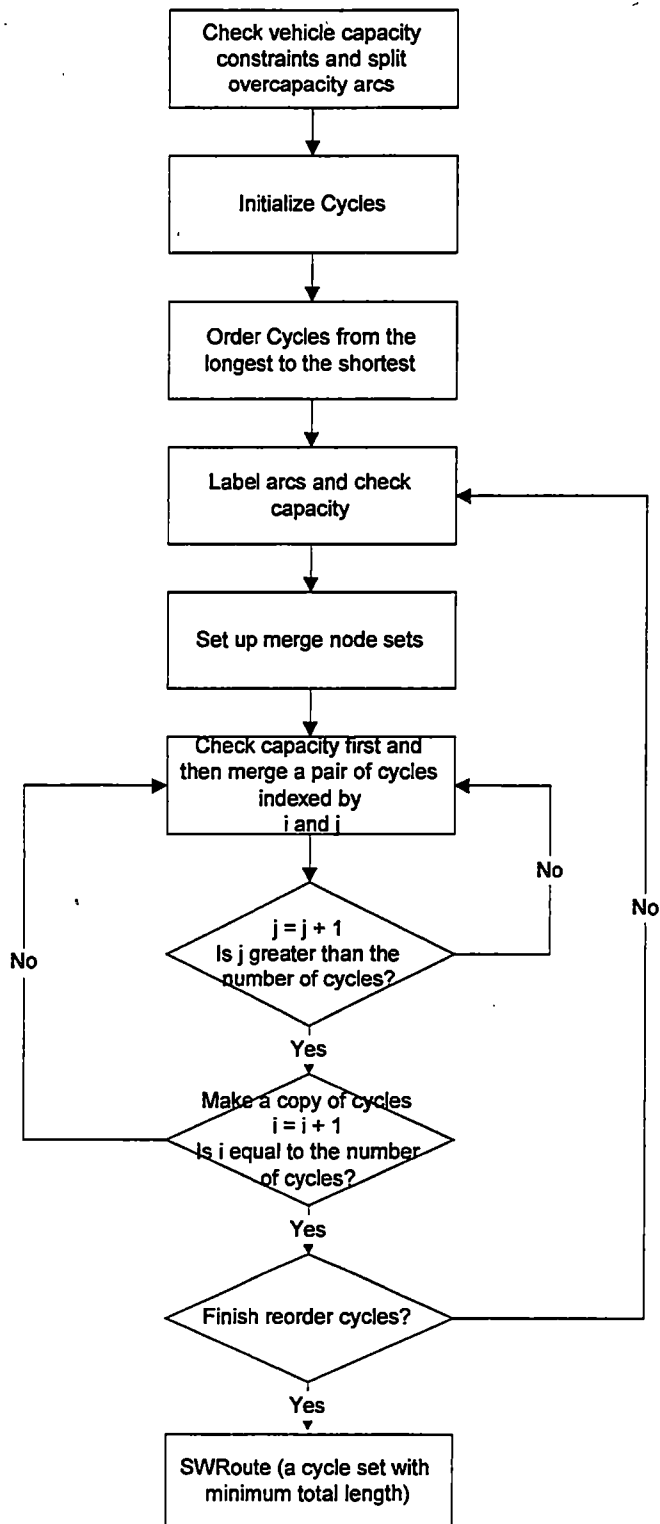


Figure 3.1 A Flow Diagram of the Augment-Merge Algorithm

3.2.3 An Example of the Augment-Merge Algorithm

Consider a CARP network depicted in Figure 3.2 with all arc length and demand as indicated¹⁰. It is assumed that the depot is node 1 and vehicle capacity is 12

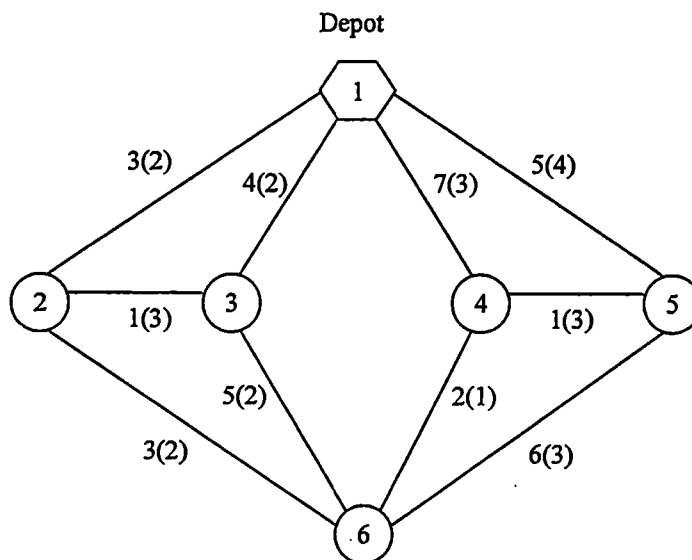


Figure 3.2 A CARP Network

Step 1: No arcs are found to violate the vehicle capacity constraint.

Step 2 and Step 3: For the undirected network in Figure 3.2, ten cycles are constructed by finding two shortest paths from each endpoint of each link to depot node 1 plus the link itself. The cycle for link (5, 6) is composed of a shortest path from node 1 to node 5 and the other shortest path from 6 to 1. So, that cycle is 1 5 6 2 1 and (5, 6) is the key link, which is underlined. Ten cycles are ordered by length in the following table.

Since no two cycles are same, no cycles are discarded in this network.

¹⁰ 3(2): 3 indicates the length of the arc and 2 indicates the demand on the same arc.

<u>Cycle Number</u>	<u>Cycle Length</u>	<u>Cycle Demand</u>	<u>Cycle</u>
1	17	3	1 <u>5 6</u> 2 1
2	15	2	1 <u>2 6</u> 3 1
3	15	2	1 <u>3 6</u> 2 1
4	14	1	1 5 <u>4 6</u> 2 1
5	13	3	<u>1 4</u> 5 1
6	12	3	1 <u>5 4</u> 5 1
7	10	4	<u>1 5</u> 1
8	8	2	<u>1 3</u> 1
9	8	3	1 <u>2 3</u> 1
10	6	2	<u>1 2</u> 1
Total	118	25	

Step 4. First, label cycles from the longest cycle 1 to the shortest cycle 10 and underline those arcs to be traversed. Arcs included in a longer cycle always have a high priority to be traversed when the cycle is compared with a shorter cycle. In the example, the sum of demands on all links in cycle 1 does not exceed the capacity, so all links on cycle 1 can be serviced. The same principle is applied to other cycles. However, since link (1, 5), (5, 6), (6, 2), and (2, 1) are served by cycle 1, they are not to be served by other cycles and their reversed link (5, 1), (6, 5), (2, 6), and (1, 2) are not to be served by other cycles either.

Cycle Number	Cycle Length	Cycle Demand	Cycle
1	17	11	<u>1 5 6 2 1</u>
2	15	4	1 2 <u>6 3 1</u>
3	15	0	1 3 6 2 1
4	14	4	1 <u>5 4 6</u> 2 1
5	13	3	<u>1 4</u> 5 1
7	10	0	1 5 1
8	8	0	1 3 1
9	8	3	1 <u>2 3</u> 1
10	6	0	1 2 1
Total	118	25	

Next, discard those cycles not underlined. They are cycle 3, cycle 6, cycle 7, cycle 8, and cycle 10. The result is as below.

Cycle Number	Cycle Length	Cycle Demand	Cycle
1	17	11	<u>1 5 6 2 1</u>
2	15	4	1 2 <u>6 3 1</u>
4	14	4	1 <u>5 4 6</u> 2 1
5	13	3	<u>1 4</u> 5 1
9	8	3	1 <u>2 3</u> 1
Total	67	25	

Step 5. Identify head-merging nodes and tail-merging nodes. They are nodes on arcs without underlined parts, i.e., the arcs with zero demand. The depot node is not counted as a head-merging node or a tail-merging node. As a result, the network in Figure 3.2 has following head-merging and tail-merging node sets.

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
1	17	11	<u>1 5 6 2 1</u>	No	No
2	15	4	1 2 <u>6 3 1</u>	2, 6	No
4	14	4	1 <u>5 4 6 2 1</u>	5	6, 2
5	13	3	<u>1 4</u> 5 1	No	4, 5
9	8	3	1 <u>2 3</u> 1	2	3
Total	67	25			

Step 6, Step 7 and Step 8: Omit cycle 1 because it has no head-merging nodes nor tail-merging nodes. Also, omit cycle 2 because it has no tail-merging nodes. Consider cycle 4 and cycle 5, then cycle 4 and reversed cycle 5 (Cycle 1 5 4 1 with head-merging nodes 5, 4 and no tail-merging nodes). Always compare head-merging nodes from cycle 4 and tail-merging nodes from cycle 5 and its reverse cycle. Since no common nodes are found, continue to compare cycle 4 and cycle 9 and merge cycle 4 and cycle 9 at node 2.

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
1	17	11	<u>1 5 6 2 1</u>	No	No
2	15	4	1 2 <u>6 3 1</u>	2, 6	No
5	13	3	<u>1 4 5 1</u>	No	4, 5
4-9	16	7	1 <u>5 4 6 2 3 1</u>	5	3
Total	61	25			

Furthermore, compare cycle 5 and cycle 4-9 and merge them at node 5. The merged cycle has demand 10, which is lower than the vehicle capacity 12, so the merging can be implemented.

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
1	17	11	<u>1 5 6 2 1</u>	No	No
2	15	4	1 2 <u>6 3 1</u>	2, 6	No
4-5-9	19	10	<u>1 4 5 4 6 2 3 1</u>	No	3
Total	51	25			

Step 9: On the base of cycles in Step 4, change the order of cycle 1 and cycle 2, repeat Step 6-Step8. It is found that there does not exist cycles for merging.

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
2	15	4	1 2 <u>6 3 1</u>	2, 6	No
1	17	11	<u>1 5 6 2 1</u>	No	No
4	14	4	1 <u>5 4 6 2 1</u>	5	6, 2
5	13	3	<u>1 4 5 1</u>	No	4, 5
9	8	3	1 <u>2 3 1</u>	2	3
Total	67	25			

Continuously, based on the generated cycles in Step 5, change the order of cycle 1 and cycle 4, cycle 1 and cycle 5, repeat Step 6-Step 8.

When it comes to the order change between cycle 1 and cycle 4, the following merging result is found.

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
4	14	4	1 <u>5 4 6 2 1</u>	5	6, 2
1	17	11	<u>1 5 6 2 1</u>	No	No
2	15	4	1 2 <u>6 3 1</u>	2, 6	No
5	13	3	<u>1 4 5 1</u>	No	4, 5
9	8	3	1 <u>2 3 1</u>	2	3
Total	67	25			

Merging could happen between cycle 4 and cycle 2 at node 2 and node 6 and between cycle 4 and cycle 9 at node 2.

Cycles with the longest savings by merging cycle 4 and cycle 2 at node 6 are shown as below:

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
4-2	17	8	<u>1 5 4 6 3 1</u>	5	No
1	17	11	<u>1 5 6 2 1</u>	No	No
5	13	3	<u>1 4 5 1</u>	No	4, 5
9	8	3	<u>1 2 3 1</u>	2	3
Total	55	25			

The order change of cycle 1 and cycle 5 shows that a merging node 5 is found between cycle 4 and cycle 5.

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
5	13	3	<u>1 4 5 1</u>	No	4, 5
1	17	11	<u>1 5 6 2 1</u>	No	No
2	15	4	<u>1 2 6 3 1</u>	2, 6	No
4	14	4	<u>1 5 4 6 2 1</u>	5	6, 2
9	8	3	<u>1 2 3 1</u>	2	3
Total	67	25			

Two continuous mergings are shown as below.

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
5-4	17	3	<u>1 4 5 4 6 2 1</u>	No	6, 2
1	17	11	<u>1 5 6 2 1</u>	No	No
2	15	4	1 2 <u>6 3 1</u>	2, 6	No
9	8	3	1 <u>2 3 1</u>	2	3
Total	57	25			

Cycle Number	Cycle Length	Cycle Demand	Cycle	Head-Merge Nodes	Tail-Merge Nodes
5-4-2	20	7	<u>1 4 5 4 6 3 1</u>	No	No
1	17	11	<u>1 5 6 2 1</u>	No	No
9	8	3	1 <u>2 3 1</u>	2	3
Total	45*	25			

When cycle 1 and cycle 9 finish changing the order and no merging cycles are found, the CARP network solution can be obtained by choosing the cycles with the minimum total length, which is 45 as indicated. The solution shows that there are no cycles with its sum of demands exceeding the vehicle capacity.

3.3 CARP Lower Bound Computation

We now need to assess the quality of the heuristic solution. To do that, a lower bound optimal solution is computed. The CARP lower bound computation is based on a method proposed by Pearn (1988). The method solves the associated nonbipartite minimum cost perfect matching problem (minimal cost 1-matching problem) and uses the resulting matching to compute the lower bounds. Appendix B presents the definition and terms about an undirected matching network, nonbipartite minimum cost perfect matching problem and its primal-dual blossom algorithm. Section 3.3.1 describes Pearn's algorithm and section 3.3.2 presents an example to clarify the procedures of implementing the algorithm.

3.3.1 Algorithm Statement and Implementation

Computing the CARP lower bound includes two phases. Figure 3.3 shows iterating steps of computing lower bounds in a flow diagram.

Phase I: Matching

Step 1: Calculate $M = \lceil Q/W \rceil$

M is the minimum number of cycles required in the CARP solution. It is derived from simply dividing the total arc demand by the vehicle capacity.

Step 2: Solve the NMCPMP Problem

Define a new set of nodes $N'(P) = \{n'(1), n'(2), \dots, n'(P)\}$, where $P = 0, 2, 4, \dots, r$ an even number no greater than r (the number of odd-degree nodes from N'). $N'(0)$, $N'(2)$, ..., and $N'(P-2)$, $N'(P)$ are sets of pairs of matching nodes in $G'(N', E', C')$.

$N'(2) = \{n'(1), n'(2)\}$ and $N'(2)$ includes one pair of matching nodes, which are $n'(1)$ and $n'(2)$.

Step 3: Order a Set of New Edges

For every node $n(i)$ in the original network node set N and the depot node X_0 , the shortest path links between $n(i)$ and X_0 , $SPL(n(i), X_0)$, are formed. $D(R, i)$ is the number of links incident to node $n(i)$ and R is a set of links with nonzero demand. According to the length of the shortest path, order $SPL(n(i), X_0)$ and list $D(R, i)$ for all nodes.

Phase II Node Scanning

Step 4: Add New Edges and Calculate Artificial Incident Link Set $I(P)$

Define $E'(P)$ as new edge sets represented by the matching solution. Its edge costs are defined as follows: For $n'(i), n'(j)$ in N' , $e'(n'(i), n'(j)) = SPL(n'(i), n'(j))$ and $e'(n'(i), n'(j)) \in E'(P)$; For $n'(i)$ in N' and the depot node X_0 , $e'(n'(i), X_0) = SPL(n'(i), X_0)$ and $e'(n'(i), X_0) \in E'(P)$.

Initialize $P = 0$, meaning that no new edges $E'(P)$ between pairs of matching nodes pass through depot node X_0 but there are new edges added along shortest paths along matching nodes. More clearly, if $P = 2$, there are one pair of matching nodes passing through depot node X_0 and new edges $E'(P)$ includes links along $SPL(n(i), X_0)$. Let $R(P) = R \cup E'(P)$, which is formed by adding $E'(P)$ to R , so $I(P) = 2M - D(R(P), X_0)$ is generated and defined as artificial paths incident to the depot.

Step 5: Calculate $L(P)$ and $D(R, L(P))$

Let $L(P) = \text{Min}\{K \mid \sum D(R,i), i=1, \dots, K\}$ and guarantee $I(P) = \sum D(R, i), i = 1, \dots, L(P)$.

Step 6: Calculate Possible Lower Bounds $LB(P)$

Let $LB(P) = |R| + |E'(P)| + |\sum SPL(X_0, i)D(R, i), i = 1, \dots, L(P)$, which is the sum of edges between matching nodes and all possible routes coming through the depot node.

Step 7: Iterate P

Repeat Step 3-6 for $P = 0, 2, \dots, r$.

Step 8: Obtaining an Optimal Lower Bound

Compute $\text{Min}\{LB(P)\}$, for $P = 0, 2, \dots, r$. In other words, the value of the CARP lower bound is computed as the minimum of all possible lower bounds $LB(P)$, which is $\text{Min}\{LB(P)\}$ over P .

3.3.2 An Example of Lower Bound Computation for Figure 3.2 Network

Appendix B discusses the procedures of deriving a matching network from Figure 3.2 network. The matching network is shown as Figure B.1 in Appendix B. The result of the primal-dual blossom algorithm shows that nodes(2, 3) and nodes(4, 5) (see Appendix B) are two pairs of matching nodes. These two pairs of matching nodes are used for lower bound computation.

Step 1: The total demand on the network $Q = 25$ units. Also, it is supposed that the vehicle capacity $W = 12$. Therefore, $M = 3$.

Step 2: $r = 4$ and $P = 0, 2, 4$. $N'(4) = \{\text{node 2, node 3, node 4, node 5}\}$.

Step 3: Shortest paths between all nodes and depot node 1 is calculated and ordered as below. $i = 2, 3, 4, 5, 6$.

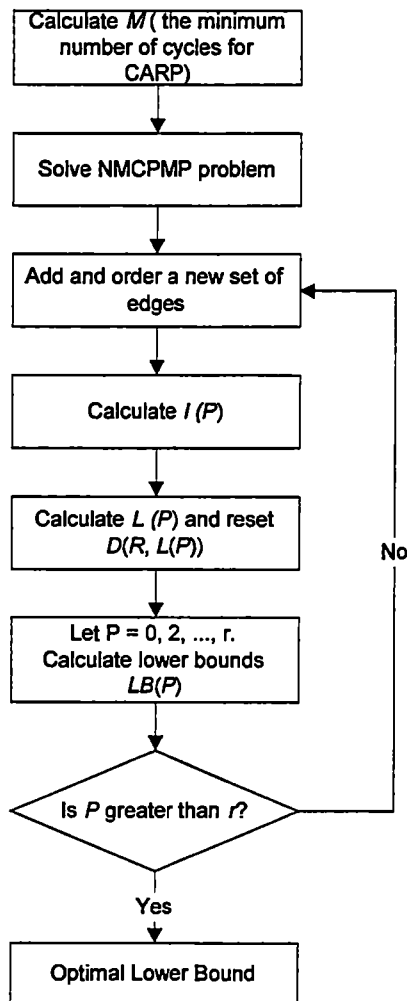


Figure 3.3 A Flow Diagram of Computing CARP Lower Bound

Node i	2	3	5	6	4
$SPL(i, 1)$	3	4	5	6	6
$D(R, i)$	3	3	3	4	3

Step 4: Let $P = 0$. The network becomes Figure 3.4 and two new edges between matching nodes 2 and 3 and nodes 4 and 5 are added.

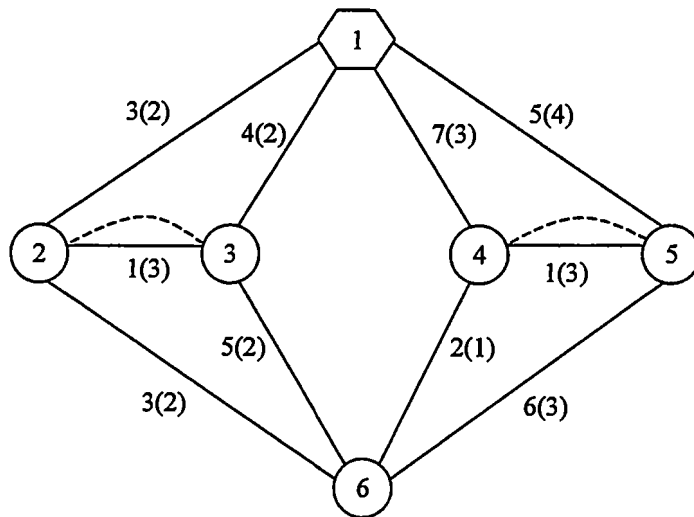


Figure 3.4 A Network when $P = 0$

$$R(0) = R \cup e'(2,3) \cup e'(4,5).$$

$$D(R(0), 1) = 4.$$

$$I(0) = 2M - D(R(0), 1) = 2 * 3 - 4 = 2$$

Step 5. $L(0) = \text{Min. } \{k \mid D(R, 2) (k = 1),$

$$D(R, 2) + D(R, 3) (k = 2),$$

$$D(R, 2) + D(R, 3) + D(R, 5) (k = 3),$$

$$\begin{aligned}
& D(R, 2) + D(R, 3) + D(R, 5) + D(R, 6) \quad (k = 4), \\
& D(R, 2) + D(R, 3) + D(R, 5) + D(R, 6) + D(R, 4) \quad (k = 5) \\
& = \text{Min. } \{1, 2, 3, 4, 5\} \\
& = 1
\end{aligned}$$

Node <i>i</i>	2	3	5	6	4
<i>SPL</i> (<i>i</i> , 1)	3	4	5	6	6
<i>D</i> (<i>R</i> , <i>i</i>)	2	3	3	4	3

Step 6. The total edge length with nonzero demand $|R| = 37$.

$$\begin{aligned}
LB(0) &= |R| + C_{23} + C_{45} + D(R, 2) * SPL(2, 1) \\
&= 37 + 1 + 1 + 2 * 3 \\
&= 45
\end{aligned}$$

Step 7. Similarly, let $P = 2$, the network will be changed in two ways. Figure 3.5 and Figure 3.6 reflect these two changes.

Case 1.

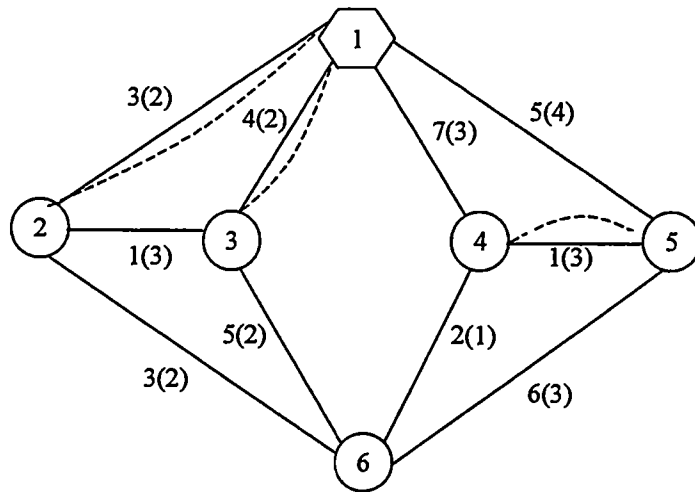


Figure 3.5 A Network when $P = 2$ in Case 1

$$R(2) = R \cup e'(1,2) \cup e'(1,3)$$

$$D(R(2), 1) = 6$$

$$I(2) = 2M - D(R(2), 1) = 2 * 3 - 6 = 0$$

$$L(2) = 0$$

$$LB(2) = |R| + C_{12} + C_{13} + C_{45}$$

$$= 37 + 3 + 4 + 1$$

$$= 45$$

Case 2.

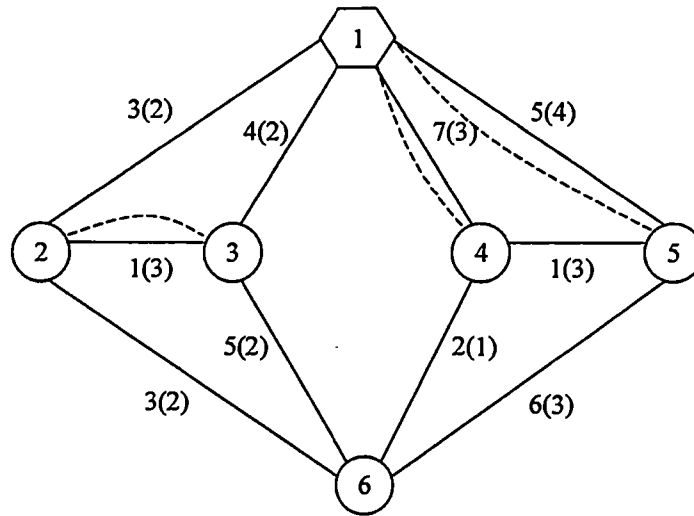


Figure 3.6 A Network when $P = 2$ in Case 2

$$R(2) = R \cup e'(1,4) \cup e'(1,5)$$

$$D(R(2), 1) = 6$$

$$I(2) = 2M - D(R(2), 1) = 2 * 3 - 6 = 0$$

$$L(2) = 0$$

$$LB(2) = |R| + C_{23} + C_{14} + C_{15}$$

$$= 37 + 1 + 7 + 5$$

$$= 50$$

Let $P = 4$, the network becomes Figure 3.7:

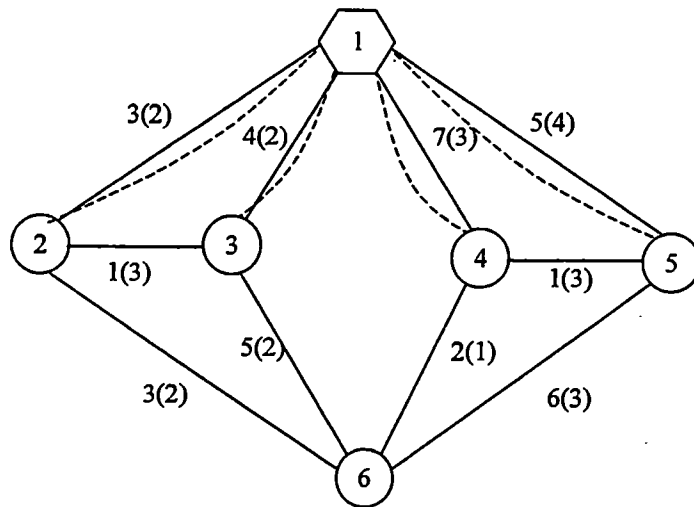


Figure 3.7 A Network when $P = 4$

$$R(4) = R \cup e'(1,4) \cup e'(1,5) \cup e'(1,2) \cup e'(1,3)$$

$$D(R(4), 1) = 8$$

$$I(4) = 2M - D(R(4), 1) = 2 * 3 - 8 = -2$$

$$L(4) = 0$$

$$LB(4) = |R| + C_{12} + C_{13} + C_{14} + C_{15}$$

$$= 37 + 3 + 4 + 7 + 5$$

$$= 56$$

$$\text{Step 8. } LB = \text{Min. } \{LB(0), LB(2), LB(4)\} = \{45, 45, 50, 56\} = 45$$

Therefore, the lower bound for the CARP network is 45 units. The CARP result is also shown as 45.

3.4 Data Structures and Their Implementation for Solving CARP and NMCPMP

The algorithms applied for solving CARP and NMCPMP require the manipulation of data, particularly data sets representing arc and node information, or representing trees or other network structures. Data structures are built outside of the GIS system. This points out the need for different representation of spatial objects for different purposes in an SDSS. The use of data structures has a considerable effect on the algorithmic performance of SWRoute models. Major data structures implemented for the thesis are doubly linked lists, disjoint sets, d -heaps, and queues. The augment-merge algorithm mainly makes use of doubly linked lists, and the primal-dual blossom algorithm is implemented by using disjoint sets. The following sections will concentrate on the doubly linked list and disjoint sets, mainly discussing how these data structures store and manipulate network information and how they are fit into the algorithms.

3.4.1 Doubly Linked Lists and CARP's Augment-Merge Algorithm

Based on a forward star network, which stores arc and node sets of network as dynamic arrays, a doubly linked list is designed as a data structure to implement operations on a list of arcs for the augment-merge algorithm. All arcs and their associated nodes in a doubly linked list take unique index numbers from the forward star network. More specifically, the list is organized with following three components.

1. three pointers to a cell indicating locations of the head, the current, and the tail of the list.

2. a pointer to another two structures which store dynamic arrays containing merging attributes, such as a list of reachable head nodes and tail nodes and their corresponding number.
3. linked list information with respect of its length, the total distance and demand of the list covered. A cell of a doubly linked list consists of three fields, data, a pointer to the previous cell, and a pointer to the next cell. A simple doubly linked list is depicted in Figure 3.8.

According to concepts of the augment-merge algorithm, the list in Figure 3.8 only includes one route. Routes of traversing all arcs in a specific network are stored in another doubly linked list. Data components of its cells are pointed to the structure of the linked list in Figure 3.9.

Doubly linked lists store the next pointer of the last cell to the first cell and the previous pointer of the first cell to the next cell. The wraparound fashion of the list makes the operation of a single network route convenient since the route starts and ends at the same depot node. Advantages of using doubly linked lists are: they allow us to traverse the list in either direction and perform insertion, and they delete and merge two lists in $O(1)$ linear running time, which is superior to singly linked lists and arrays.

3.4.2 Disjoint Sets and NWPMP's Primal-Dual Blossom Algorithm

The primal-dual blossom algorithm for solving NWPMP needs a special data structure to handle blossom shrinkings and unshrinkings. Disjoint sets are a simple and elegant data structure to implement this kind of operation and its running time is $O(1)$. Initially, all nodes in a matching network are created as root nodes in individual new sets.

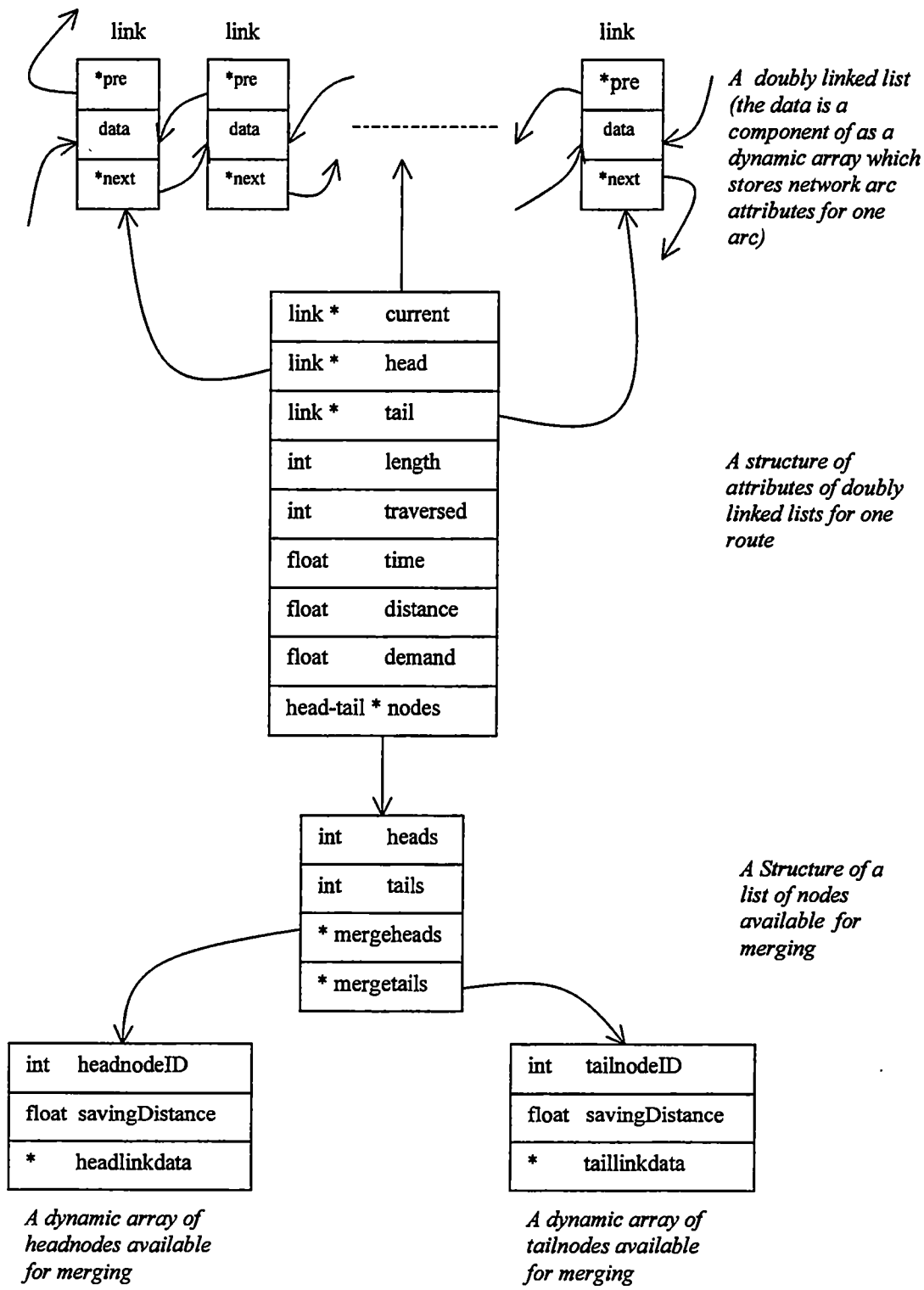


Figure 3.8 A Schematic Description of a Doubly Linked List for SWRoute

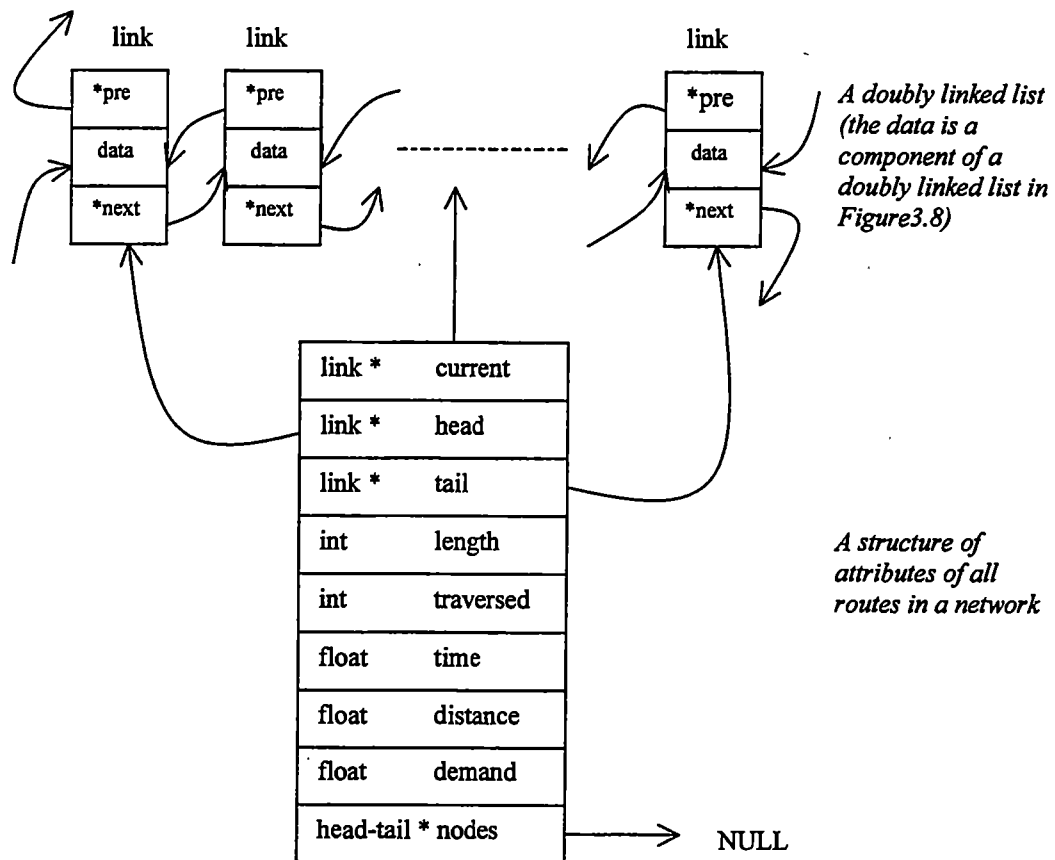


Figure 3.9 A Schematic Description of Doubly Linked Lists for all Routes in a Network

Each new set is stored as a structure that includes one pointer to its parent, the other pointer to its son in the tree, and the size of the set. All pointers from the root nodes point to themselves. The union of two sets needs to return the root node and append a smaller-size set to a larger-size set. The set attributes for all nodes within the two sets are updated and a new set is generated.

Let x, y be two nodes in the network and $x \neq y$. Three set operations are formulated as below.

1. $\text{makeset}(x)$ creates a new set containing the single element x , which previously is in no set.
2. $\text{find}(x)$ returns a root element of the set containing node x .
3. $\text{link}(x, y)$ form a new set that is the union of the two sets whose root elements are x and y after comparing with the size of the x and y sets.

In the primal-dual blossom algorithm, a dynamic array stores all matching nodes with their associated structures. One of the structure's components is the disjoint set data structure. Other components include the node index, its matching node index, its previous node index on a path, pseudonode attributes, and primal-dual variables. The disjoint set operations rely on the pointers of the previous path nodes to find a pseudonode for shrinking. Nodes and arcs in a union set become attributes of the pseudonode. The dynamic array including a disjoint set is shown in Figure 3.10.

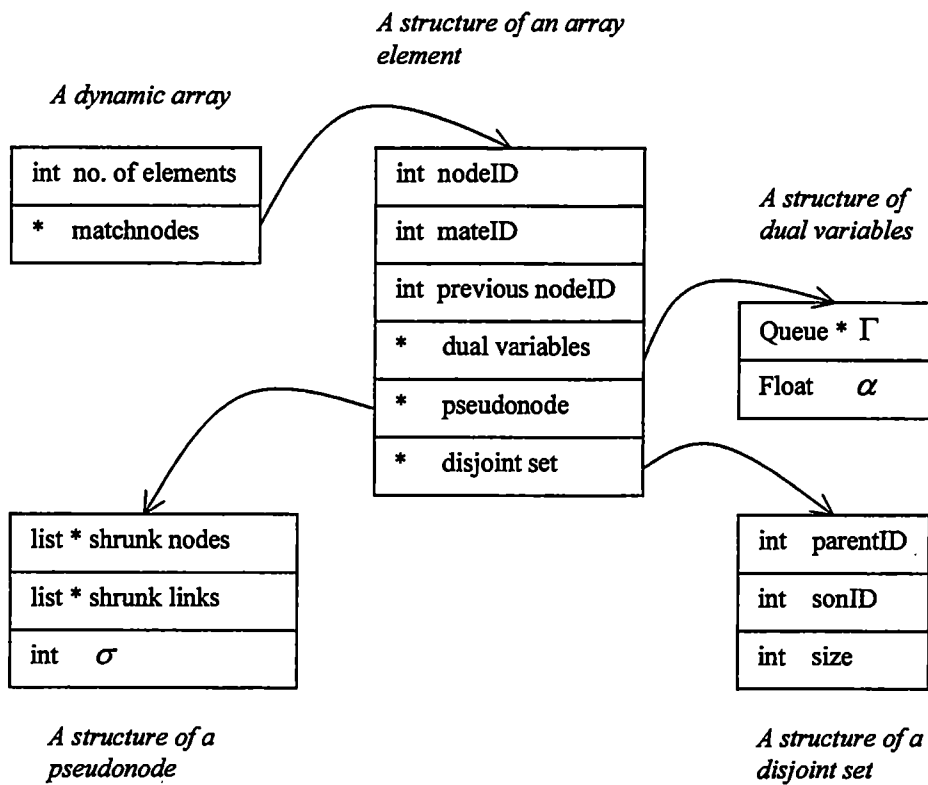


Figure 3.10 A Schematic Description of Disjoint Sets and their Related Structure

Components

3.6 Concluding Remarks

This chapter has reviewed the literatures of capacitated arc routing problems and concluded that CARP belongs to the class of *NP*-hard problems. The augment-merge algorithm proposed by Golden, *et. al.* (1983) is used as an approximate heuristic solution for a class of CARP. The CARP lower bound computation is based on a method proposed by Pearn (1988). His algorithm requires building a nonbipartite weighted matching network, which can be solved by the primal-dual blossom algorithm (Edmonds 1965, Galow 1983). The latter two algorithms have polynomial running time with less computing complexity than the augment-merge algorithm. The last part of the chapter dealt with an important aspect of SDSS. It is necessary to store, maintain, and manipulate information by using two types of data structures: those structures which facilitate mathematical modeling and algorithm implementation, and those which are used by the GIS for data maintenance and cartographic display. In this chapter, we have concentrated on the former. In the next chapter, we will discuss how these algorithms are applied for a practical use and embedded into a SWRoute GIS interface.

CHAPTER 4

SWRoute Interface Design and System Integration

4.1 Introduction

In addition to the algorithms described in the previous chapter, SWRoute is also designed to contain functions to answer "What-If?" questions about designing solid-waste collection routes. Accordingly, the SWRoute interface should integrate such functions as algorithmic solutions, database management, and user interventions into a single system with toolboxes. Generally, SWRoute consists of three sets of toolboxes: one set for partitioning a base road network into subnetworks, a second set for creating a seed node set¹¹, and a third set for generating and reporting routes. These toolboxes are accessed through menus and buttons (Figure 4.1).

As discussed in Chapter 2, the toolboxes for SWRoute are based on an object-oriented design. Figure 4.2 presents the relationships between SWRoute objects and how the toolboxes are linked with these objects. SWRoute objects and the inheritance of the objects are arranged hierarchically. The object at the highest level is SWRoute base network, whose data sources include enough information for SWRoute subnetwork generation, which can not be edited. The object at the next level is SWRoute partition network. It is derived from the SWRoute base network but with editable data sources. Along with seed node sets, SWRoute partition network serves as a template for SWRoute subnetworks. In other words, subnetworks are objects generated from a SWRoute

¹¹ Nodes where garages, landfills, and depots are located.

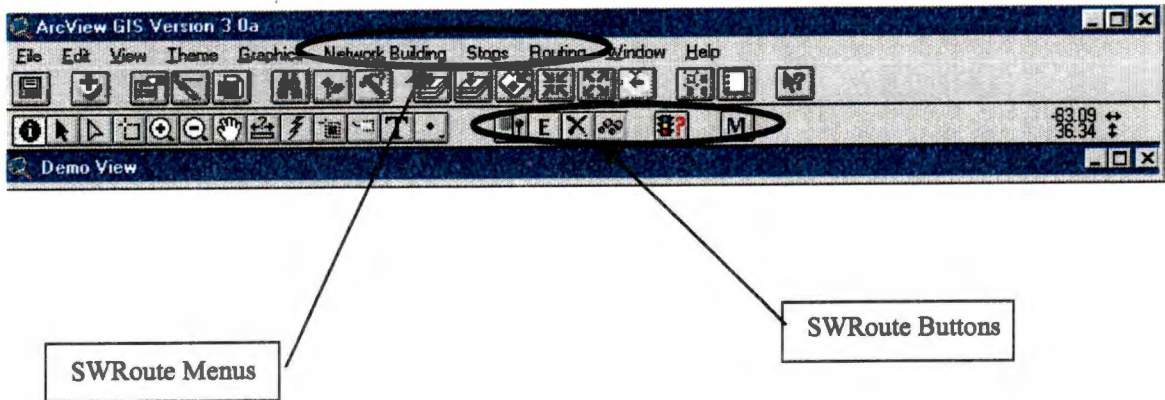


Figure 4.1 SWRoute Interface Overview

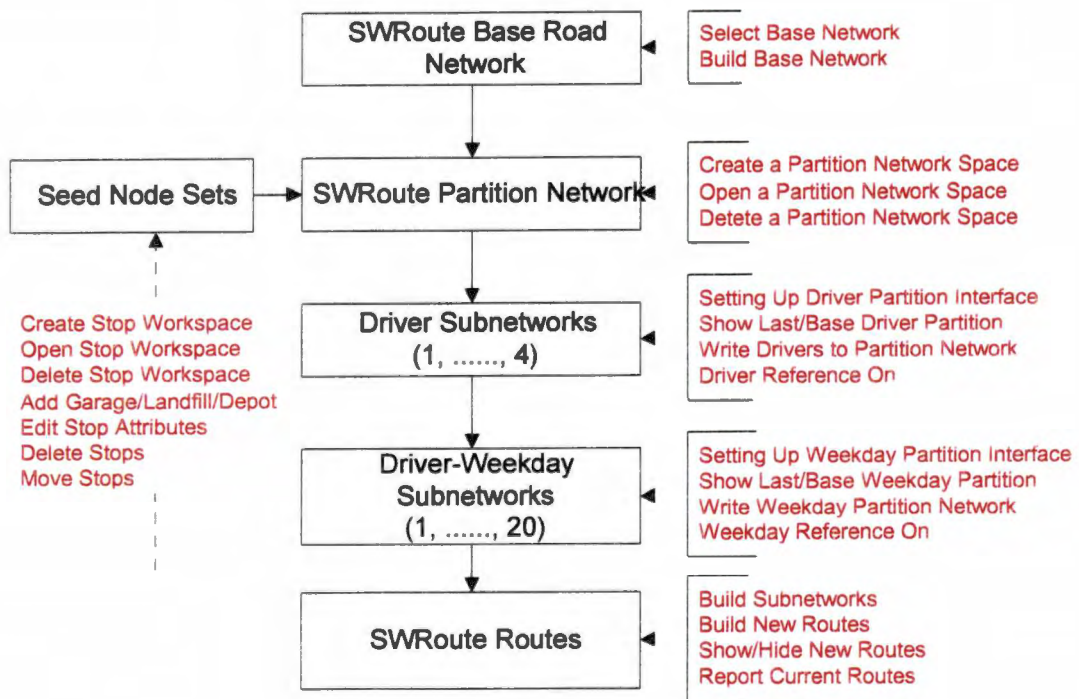


Figure 4.2 A Descriptive Flow Diagram for SWRoute Objects (Black Letters) and Their Interface Representation (Red Letters)

partition network and its seed node set. The first level of subnetwork objects is for driver partition (driver subnetworks) and the second level is for driver partitions on weekdays (driver-day subnetworks). The latter relies on the result of the former. The augment-merge algorithm and the method of computing lower bound are associated with SWRoute routes objects, the lowest level of network for achieving objects used for optimal routes. The object at the lowest level is called SWRoute route objects, which are directly linked with driver-day subnetworks. It is observed from the above discussion that all SWRoute objects are associated with inherited and unique data attributes. The following sections will discuss data attributes and operations of these objects in detail.

4.2 SWRoute Base road Network

4.2.1 Generating a SWRoute Base road Network for Hamblen County

The Hamblen county SWRoute base road network is built from four different data sources. The first is 1995 U.S. Census Bureau TIGER¹² files. They are translated into ArcView shape file format through a TIGER-to-SHAPE translator. The second is a transportation network file from Tennessee Transportation department. The source file format is MicroStation DGN file format¹³ and was created in 1997 using 1996 road network data. The third is a street map for Hamblen County. It is a paper map using 1997 road network data. The fourth is a driver survey file in DBF file format, whose data were collected by drivers who serve for Hamblen SWRoute. Based on these various data

¹² Topologically Integrated Geographic Encoding and Referencing (TIGER) is a set of line coverage files released by U.S. Census Bureau.

¹³ A file format used by Bentley Systems, Incorporated. Files with the file format keep GIS and AutoCAD properties.

sources, a base road network used for Hamblen SWRoute is generated with following guidelines.

1. Shape file format: SWRoute interface is implemented in ArcView. Its shape file can be joined or linked with DBF files. Its graphic attributes and corresponding database attributes can be edited according to changes from other digital or non-digital data sources, such as paper maps and database files. MicroStation DGN files can be imported into ArcView and overlaid with shape files if they are transformed with a same projection system.
2. Approximate geographical locations: the DGN file obtained from Tennessee Transportation Department keeps more accurate road locations than the TIGER shape file. Road locations on the TIGER shape file can be modified to more closely match DGN file.
3. Accurate road names: the TIGER shape file, Hamblen 1997 street map, and driver survey file all keep one data item about road or street names. But only Hamblen 1997 street map has the most accurate street and road names, which could be used as a reference to change a name item in the TIGER shape file and the driver survey DBF file. These names are used to update the TIGER street names.
4. Connected road network: the TIGER shape file needs to be rebuilt to keep all roads and nodes of a base road network with topological relationship.
5. Attached driver information: the driver survey DBF file has items showing the road name and the amount of solid-waste collected from households on the road by a certain driver on a certain weekday. The information could be used for building initial SWRoute.

Figure 4.3 presents a process of generating a base road network and the relationship of the original TIGER shape file with other files. Figure 4.4 shows a SWRoute interface with a base road network and Figure 4.5 are menu items used for generating base networks. The original road network shape file used for Hamblen SWRoute is translated from 1995 TIGER files. The base road network is generated from it and includes road shape nodes and lines with a topological relationship and correct and useful attributes. Detailed steps for processing the road shape lines and attributes are as follows.

Step 1: Check road name

The street map and driver DBF file are compared to make sure all streets listed in the driver DBF file can be found on the 1997 street map. If not, the street name is changed to a closest one existing on the street map.

Step 2: Check road location

ArcView road shape file and MicroStation DGN files were transformed into ArcInfo coverages so that it is possible to project them to the state plane coordinate system. In ArcView, two coverages are changed back to ArcView shape files. After two shape files are overlaid, ArcView editing tools are used to add, delete, and move nodes and links in order to make sure that every road appearing as a data item in driver DBF file exists in the road shape file.

Step 3: Attach SWRoute field survey information

In ArcView, the road network shape file was permanently joined with the driver DBF file.

Step 4: Build Topological Coverage for SWRoute

In ArcInfo, the road network shape file is changed to an ArcInfo coverage again.

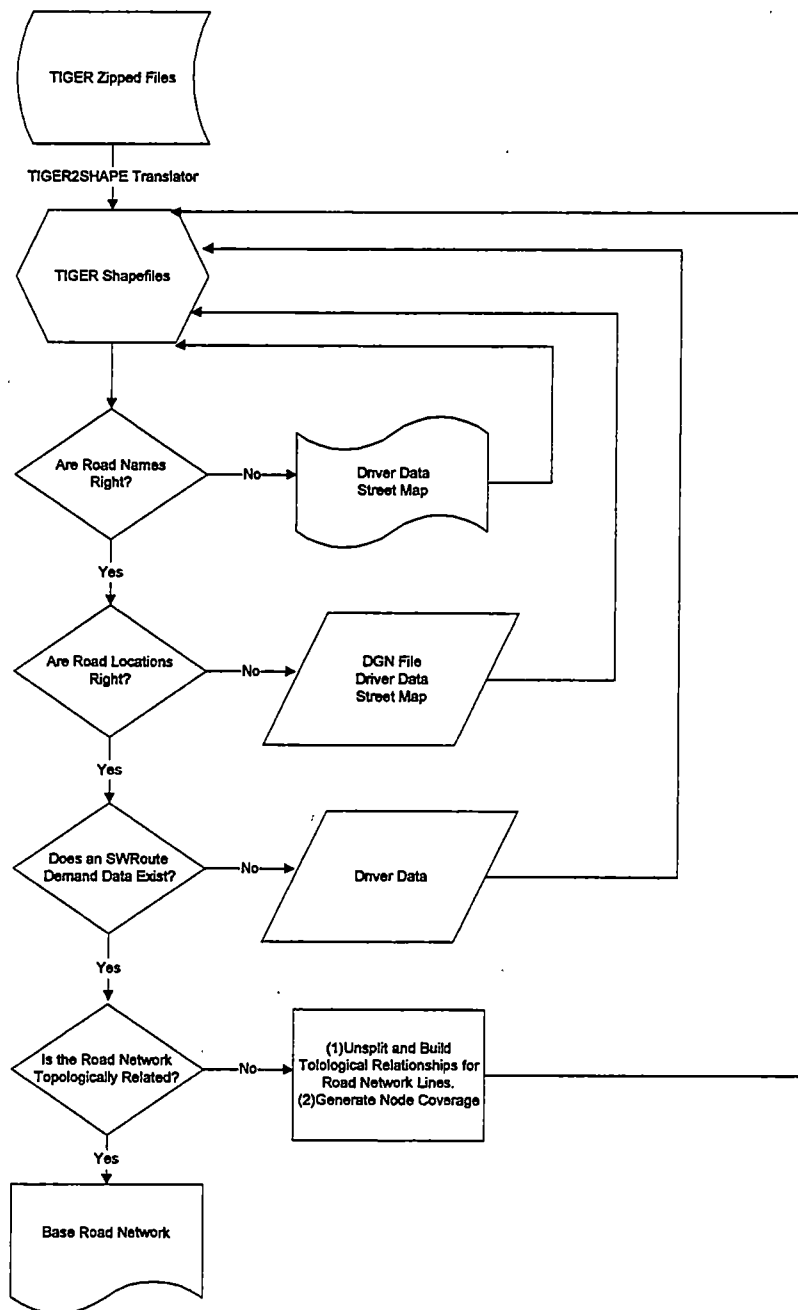


Figure 4.3 A Flow Diagram of Generating a SWRoute Base road Network

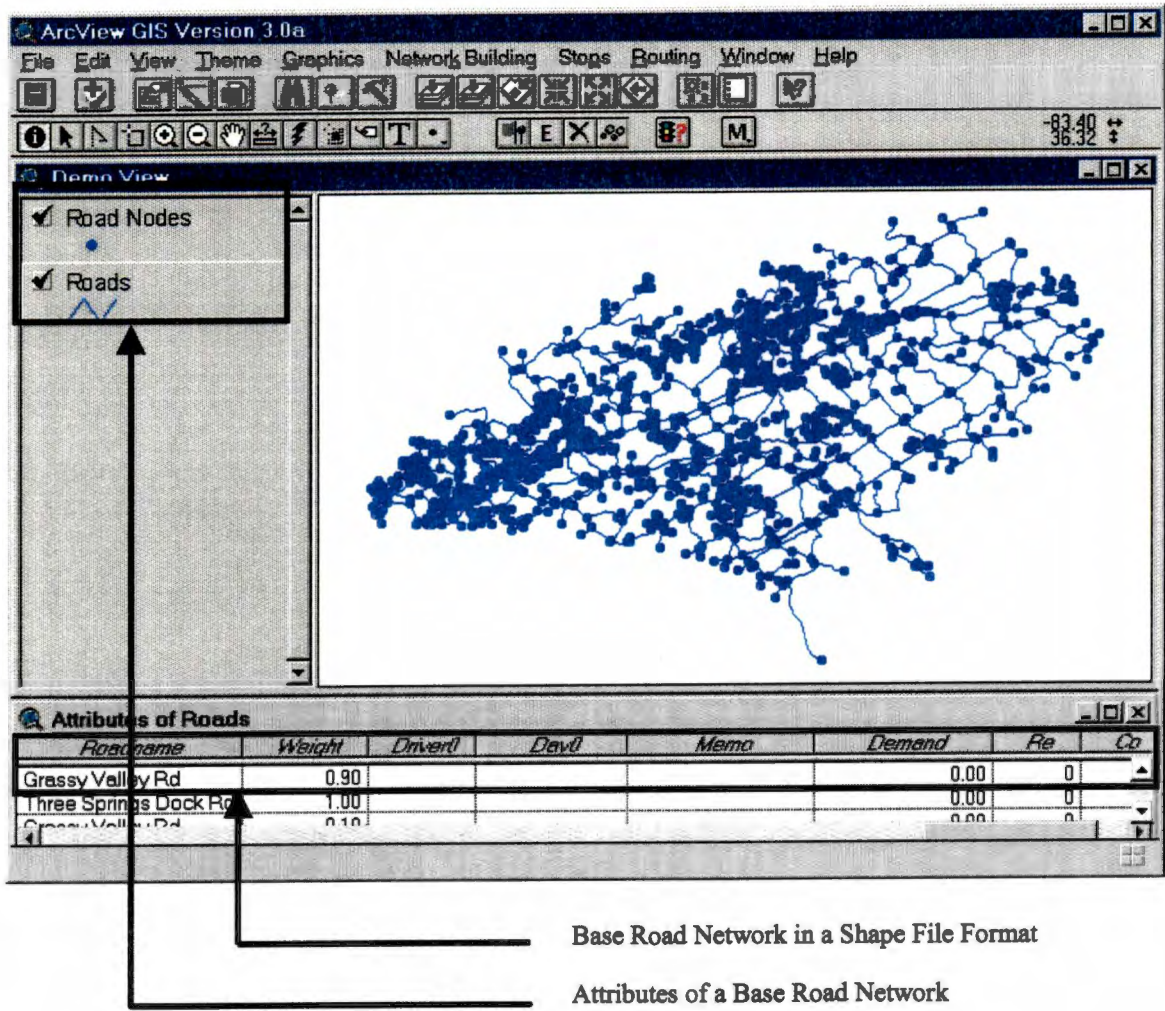


Figure 4.4 A SWRoute Base Road Network Interface



Figure 4.5 A Menu for Generating SWRoute Base road Networks

Then, topological relationships are built and a node coverage is generated from the link coverage. Finally, the “unsplit” command is used to merge all links on the same road (i.e., with the same street road name) so that the total number of links and nodes on the base road network can be reduced. The base road network is transformed from ArcInfo coverage into ArcView shape line file and a node file.

4.2.2 SWRoute Base road Network Data Attributes

The attribute table for a base road network should keep data attributes with specific data field names, which are used for building subnetworks and seed node sets. Table 4.1 lists these data attributes.

Interface functions for generating SWRoute base road networks are listed in Appendix C.

4.3 SWRoute Subnetworks

4.3.1 Generating the Hamblen County SWRoute Subnetworks

SWRoute subnetworks are defined as a set of networks for a specific driver who works on a specific weekday. Each subnetwork keeps links with non-zero demands if the links are to be traversed. If demands on links from the base road network are partitioned properly, each subnetwork should have approximately balanced total demand. In Hamblen County, there are four drivers who work five days a week. Therefore, the Hamblen County SWRoute partitions the Hamblen County into 20 subnetworks, each for one driver on one weekday.

Table 4.1 Data Attributes in a Base road Network

Data Attribute Name	Data Attribute Source	Data Attribute Function
Fnode Tnode Length Link Index Fromlong Fromlat Tolong Totat Two-way	Arcinfo coverages	Build forward star network data structure Build target network
weight	One link length divided by the total length of all links with the same name	A weight used for calculating the fraction of demand along one road
Re Co	Driver DBF file	The total number of residential and commercial stops along one road
Demand	(pounds collected per crew per household * Re + pounds collected per crew per business * Co) * weight	The amount of solid wastes generated from residential and commercial stops on one link
Driver0	Driver DBF file	The driver code representing an original driver partition
Day0	Driver DBF file	The weekday representing an original driver-weekday partition
Roadname	Fdpre + Fname + Ftype	Use as a full road name

In SWRoute, partitioning a base road network is not designed as an algorithm based approach but as GIS-based toolboxes. The toolboxes are embedded within the ArcView interface. A partitioning process can always be changed in order to achieve better partition results. Links of subnetworks from partition can be swapped between subnetworks. The following is a list of guidelines for designing the toolboxes for partitioning a base road network.

1. Intact base road network attributes: As shown in Figure 4.1, SWRoute subnetworks are directly derived from a partition network. The partition network is actually a template and a temporary work space for SWRoute subnetworks. It is generated from the SWRoute base road network. It keeps all base road network characteristics, including such attributes as topological relationships and demand.
2. Reference to other subnetworks: A partition network is used for storing subnetwork information, which are built from demand data and collected by drivers and an SWRoute subnetworks defined by a user. The drivers' SWRoute subnetworks are generated from the field survey data. The user's subnetwork is generated from the result of a network partition. Therefore, the original and the latest subnetworks are two references for generating SWRoute subnetworks. For a base road network with more than 2000 links, it is difficult to assign each link to subnetworks manually. However, it is easy to adapt and modify partitioning attributes of some links from other subnetworks in the same region and form new subnetworks.
3. Create subnetwork workspace: Generating a new partition network also creates a new workspace for storing all other files associated with a partition network and its

subnetworks. Similarly, removing a current partition network from SWRoute interface will remove everything in its directory and all associated files.

4. Stepwise partition: SWRoute first generates a set of driver subnetworks first and then driver-day subnetworks next.
5. Graphic representation of subnetworks: Different subnetworks are displayed with graphic color symbols and line weight. Legends within a view display area are used to recognize which color and line weight belongs to different subnetworks. Reference subnetworks are displayed with thematic color symbols and line weight. Thematic legends show the type of subnetworks.
6. Updated current partition status: A goal of partitioning a base road network is to obtain a set of subnetworks with approximately balanced workloads. A report function in a SWRoute interface helps keep track of the amount of workloads for driver subnetworks and driver-day subnetworks.

Figure 4.6 shows SWRoute menus for generating subnetworks. Figure 4.7 is a flow chart listing all necessary steps and their relationships for subnetwork generation.

A detailed description of generating subnetworks is as follows:

Step 1: Building a Partition Network from a Base Road Network

With one active base road network in a SWRoute interface, it is possible to either create a new partition network or open an old partition network or delete an existing partition network in SWRoute interface. There must be at least one partition network in the SWRoute interface before proceeding to the next step.

Step 2: Driver Partition

“Setting Up Driver Partition Interface” activates a group of tools with driver code

<u>C</u> reate A Partiton Network Workspace <u>O</u> pen A Partiton Network Workspace <u>D</u> elete A Partiton Network
Setting Up <u>D</u> river Partition Interface <u>S</u> how Last/Base Driver Partition Write <u>D</u> rivers To Partiton Network Driver Ref. On
Setting Up <u>W</u> eekly Partition Interface <u>S</u> how Last/Base Weekly Partition Write <u>W</u> eekly Partititon Network Day Ref. On
<u>B</u> uild Subnetworks

Figure 4.6 A Menu for Generating SWRoute Subnetworks

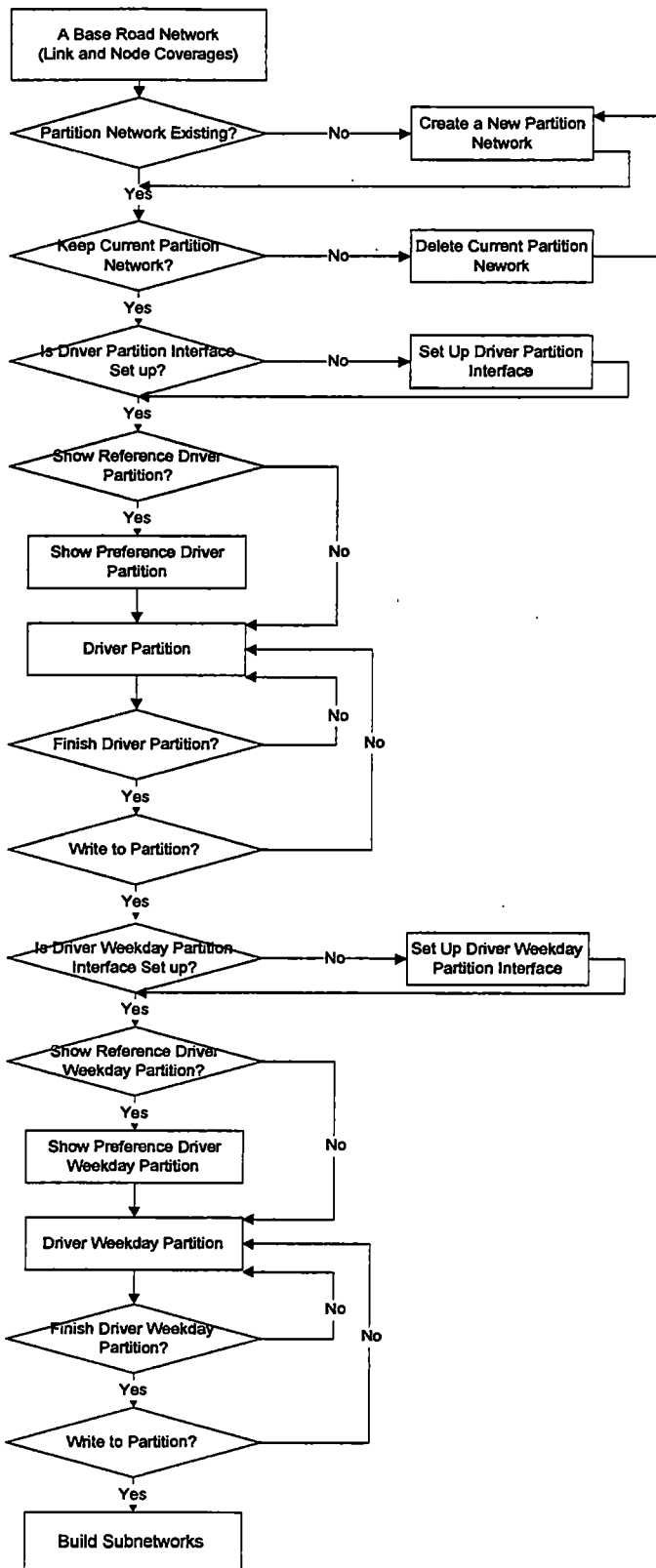


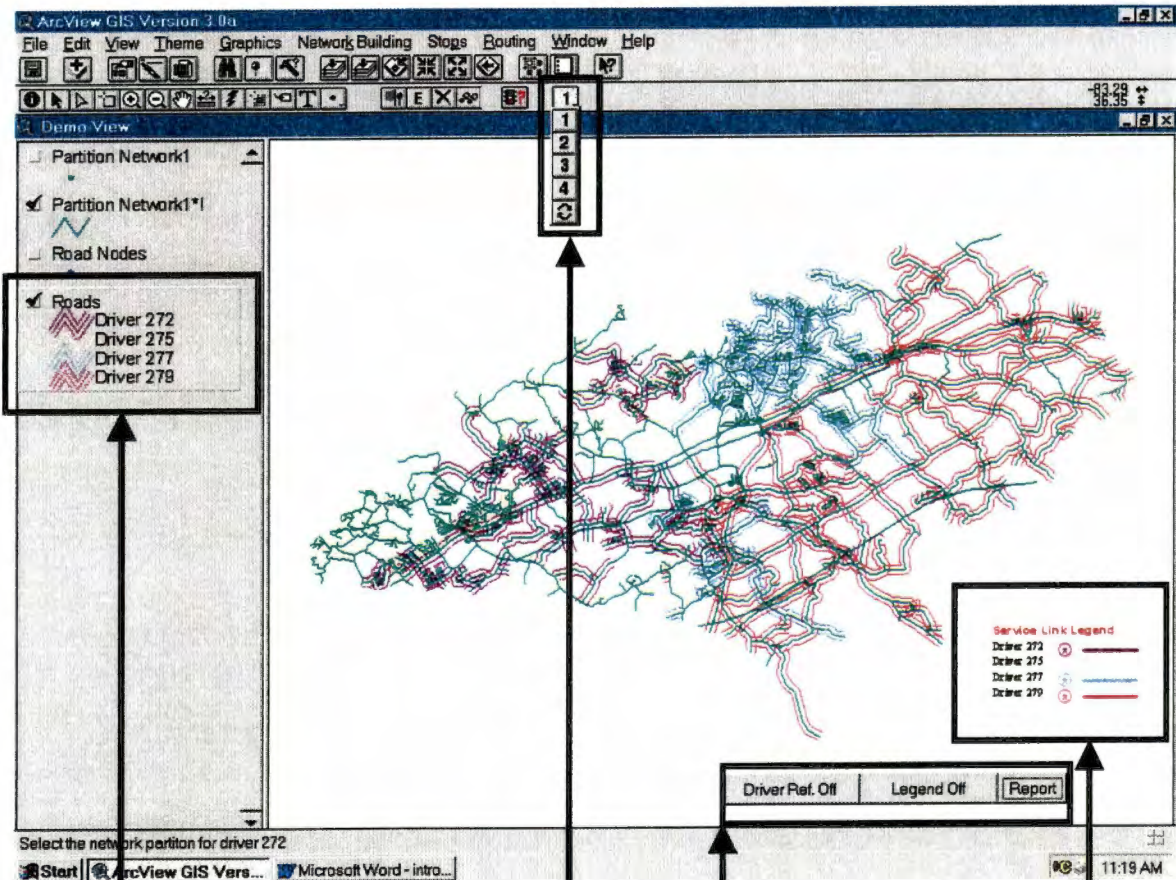
Figure 4.7 A Flow Diagram of Generating SWRoute Subnetworks

numbers along with an erase tool in the project toolbar. Three buttons labeled as “Driver Reference On/Off”, “Legend On/Off”, and “Report” are also shown in the project view area. They are used to display for driver partitions, driver partition line styles, and current partition status (Figure 4.8). A report is written as a MSOffice Word file format. It is a list of status of the current partition (Figure 4.9).

2. “Show Last/Base Driver Partition” toggles SWRoute interface menu to set up a template for SWRoute driver partition, which can be a partition result from an original base road network or the latest partition network. The partition tools are then used to either remove a partition on a link or reassign the link to a driver partition. A run-time message box shows how many links are selected for a certain driver. The report button in the project view area shows an overview partition status for all drivers (Figure 4.8).
3. Once all links with non-zero demand have been assigned to drivers, then “Write to Partition” menu choice is used to write the new partition to the attribute table of a partition network. An example for such a partition network is shown in Figure 4.10.

Step 2: Weekday Partition

1. “Setting Up Weekday Partition Interface” sets up a group of tools representing “Monday, Tuesday, Wednesday, Tuesday, and Friday” along with an erase tool in the project toolbar region. Three buttons labeled as “Day Reference On/Off”, “Legend On/Off”, “Report”, and a dropdown box listing “All Drivers, Driver272, Driver275, Driver277, Driver279” are also set up within the project view area. These buttons provide another set of tools for displaying a reference for weekday partition, driver



Showing driver subnetwork as a reference

A group of tools for driver partitioning

Buttons in the view area

Showing legends in a view area

Figure 4.8 A Driver Partition Interface after Setup

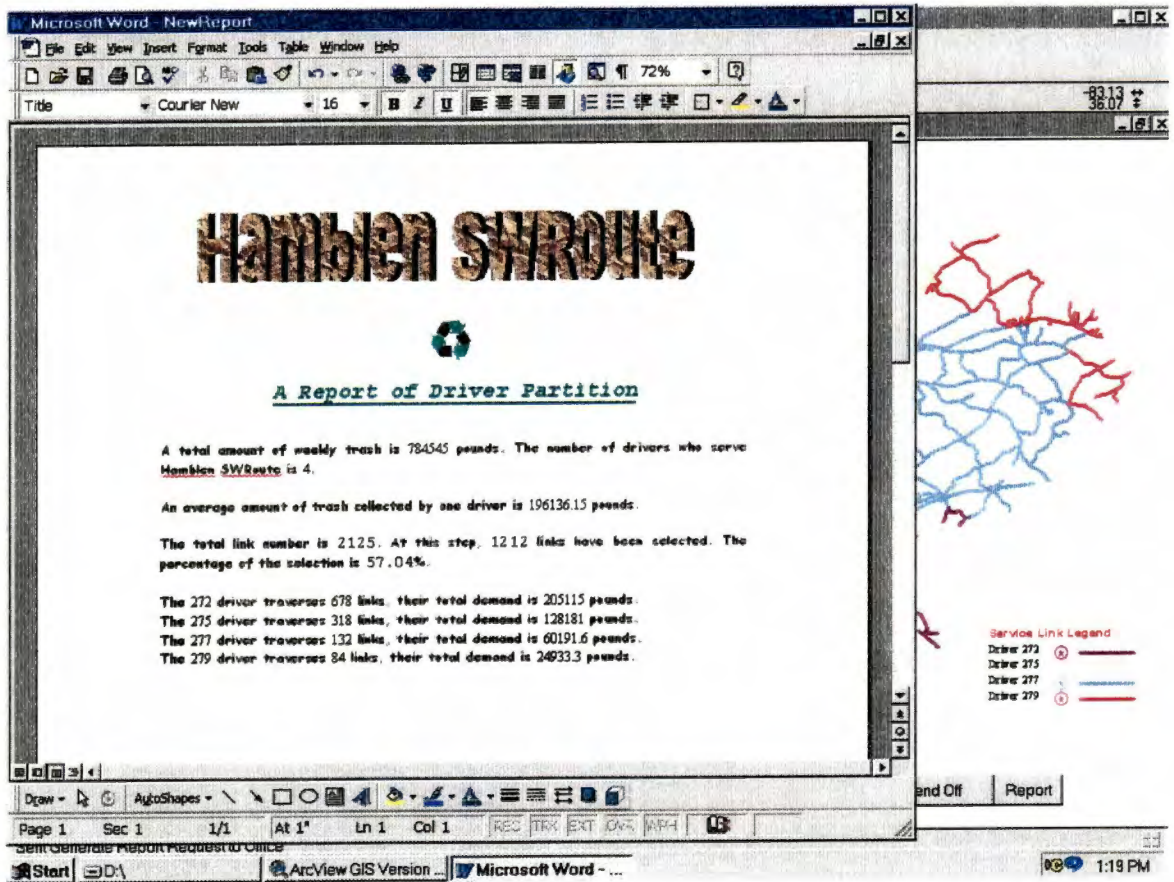


Figure 4.9 A Driver Partition Report Written in a MSOffice Document

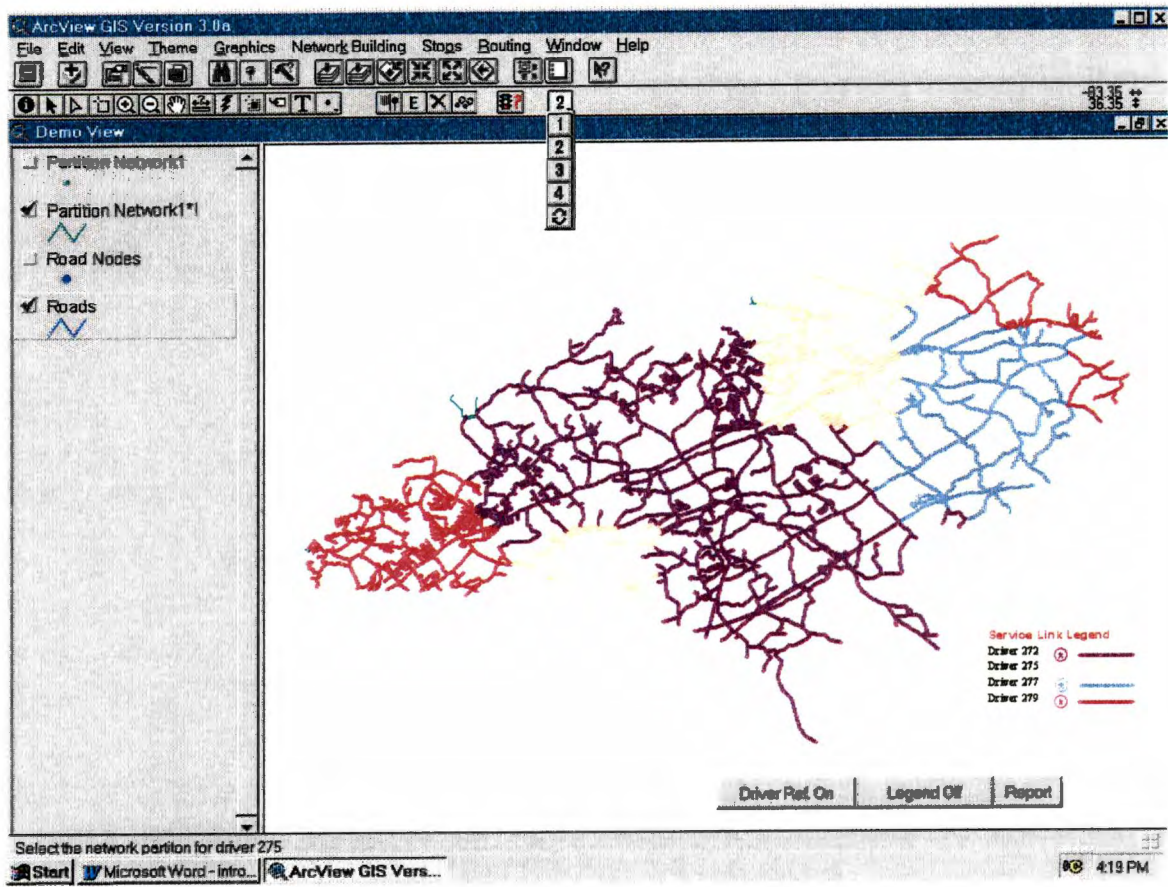


Figure 4.10 A Driver Partition Interface with Links being Partitioned

partition line styles, and current partition status (Figure 4.11). The reports about the driver-day partition are written into a notepad file format, which presents the status of the current driver-day partition (Figure 4.12). The dropdown box with a list of the driver codes is used to scale and fit maps into project view areas. Figure 4.13 presents an example how map scale and the legend change when Driver 272 is selected.

"Show Last/Base Weekday Partition" toggles SWRoute interface menu to set up a template for SWRoute driver-day partition, which can be a partition result from an original base road network partition or a latest network partition. The partition tools are then used to either remove a partition, reassign a link to a weekday partition, or directly assign a link to a weekday partition after a driver partition is finished. A run-time message box shows how many links are selected for a certain driver. The report button in the project view region shows an overview partition status for all drivers and their weekdays (Figure 4.12).

2. Once all links with non-zero demand have been assigned to driver-day combination, "Write to Partition" is used to take the above partition result as a new latest driver-day partition result, which is implemented by writing partition results to the attribute table of a partition network. An example for such a partition network is shown in Figure 4.14.

Step 3. Build Subnetworks

After all links with non-zero demand are assigned to driver and weekday, "Build Subnetworks" is applied to generate 20 forward star network files for SWRoute on the Hamblen County.

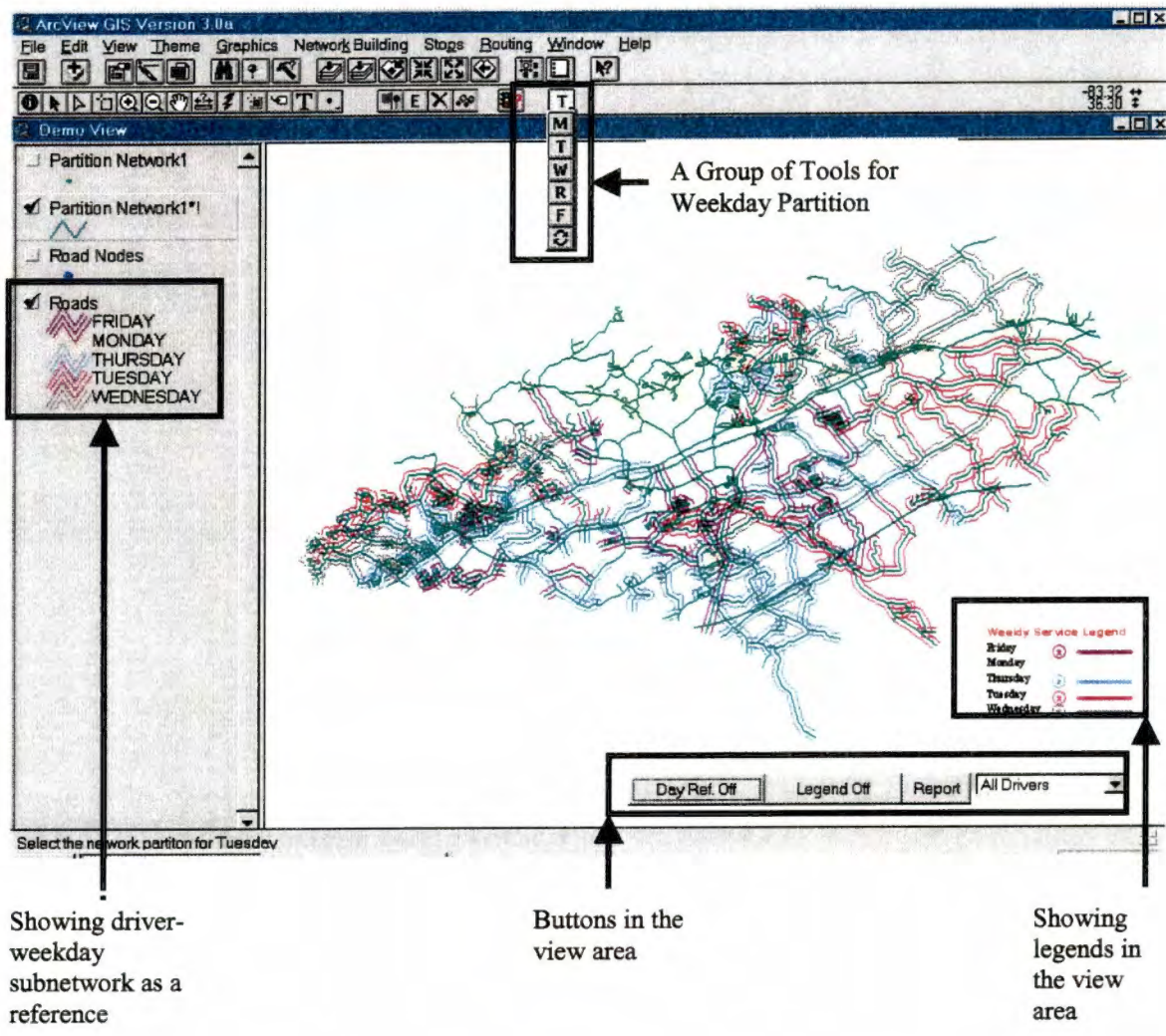


Figure 4.11 An Interface for Driver-Weekday Partition

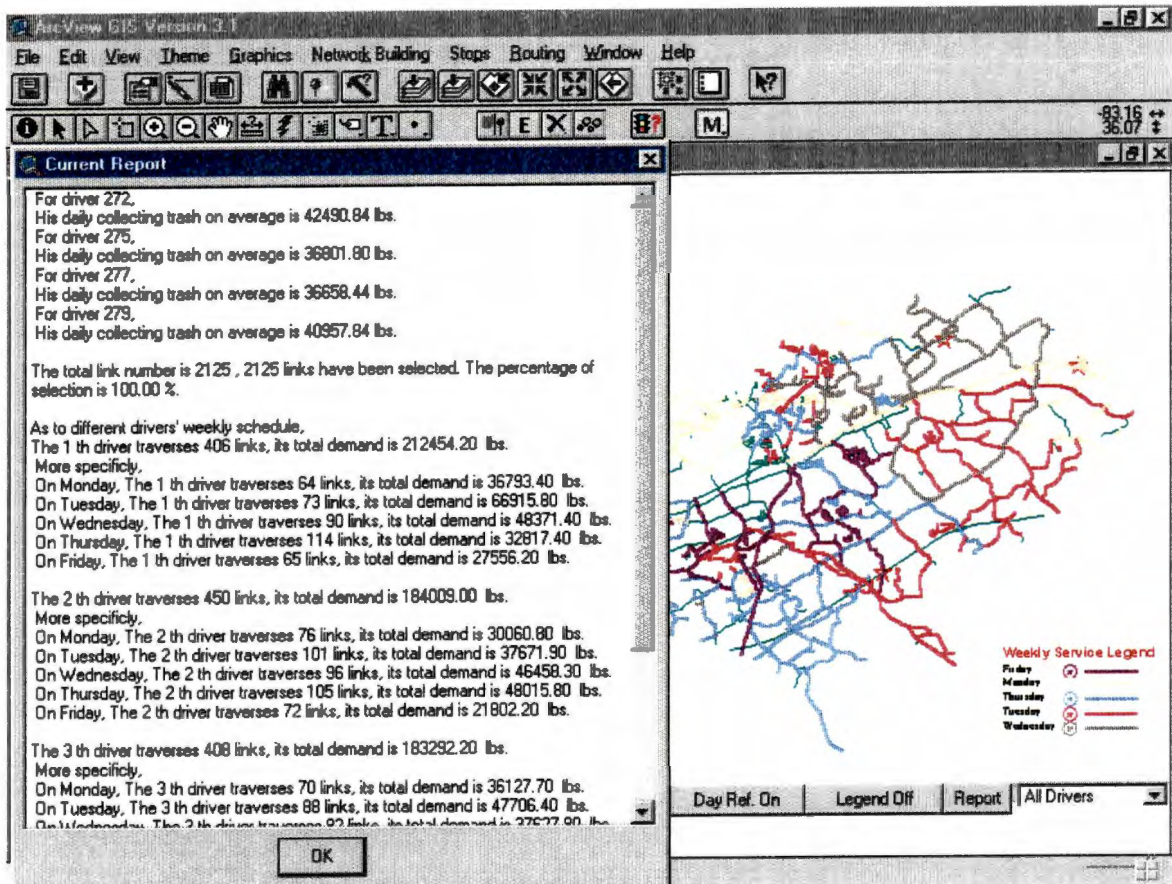


Figure 4.12 A Driver-Weekday Partition Report Generated in a Notepad

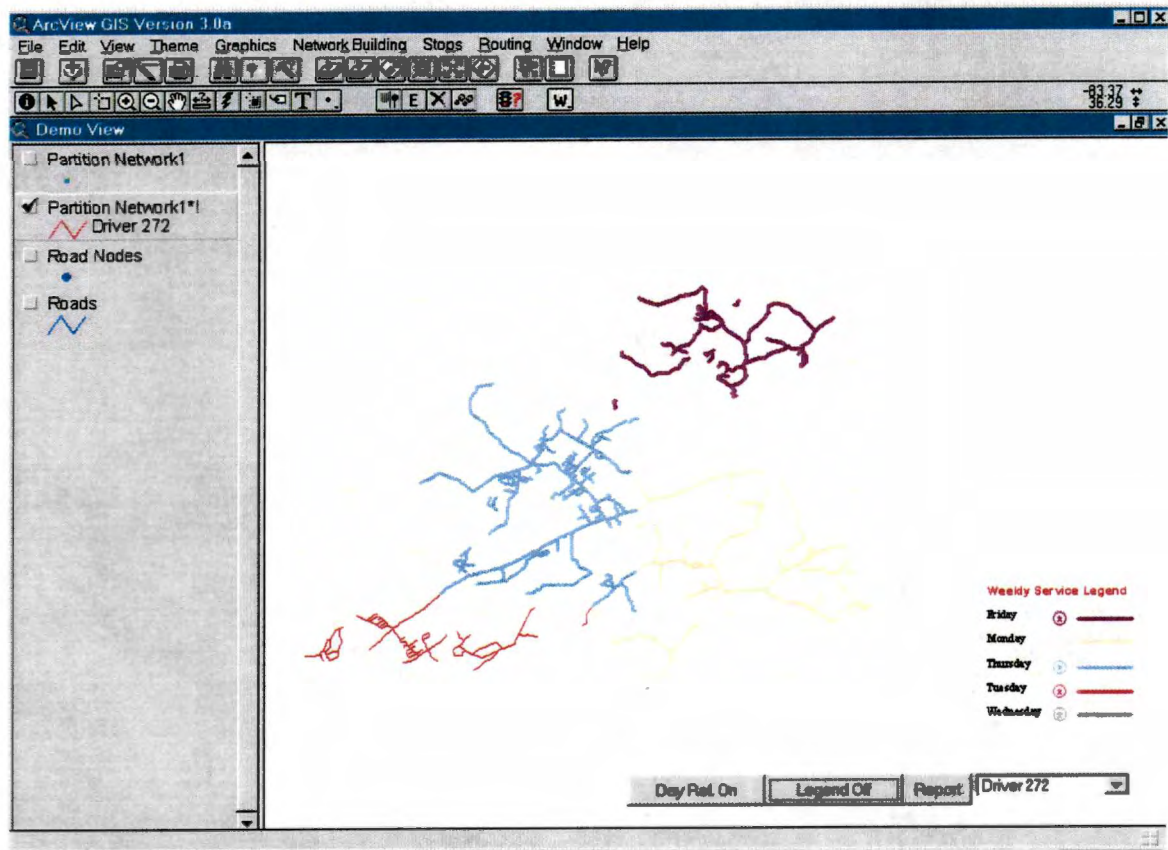


Figure 4.13 A Scaled Map for Driver 272 Subnetworks

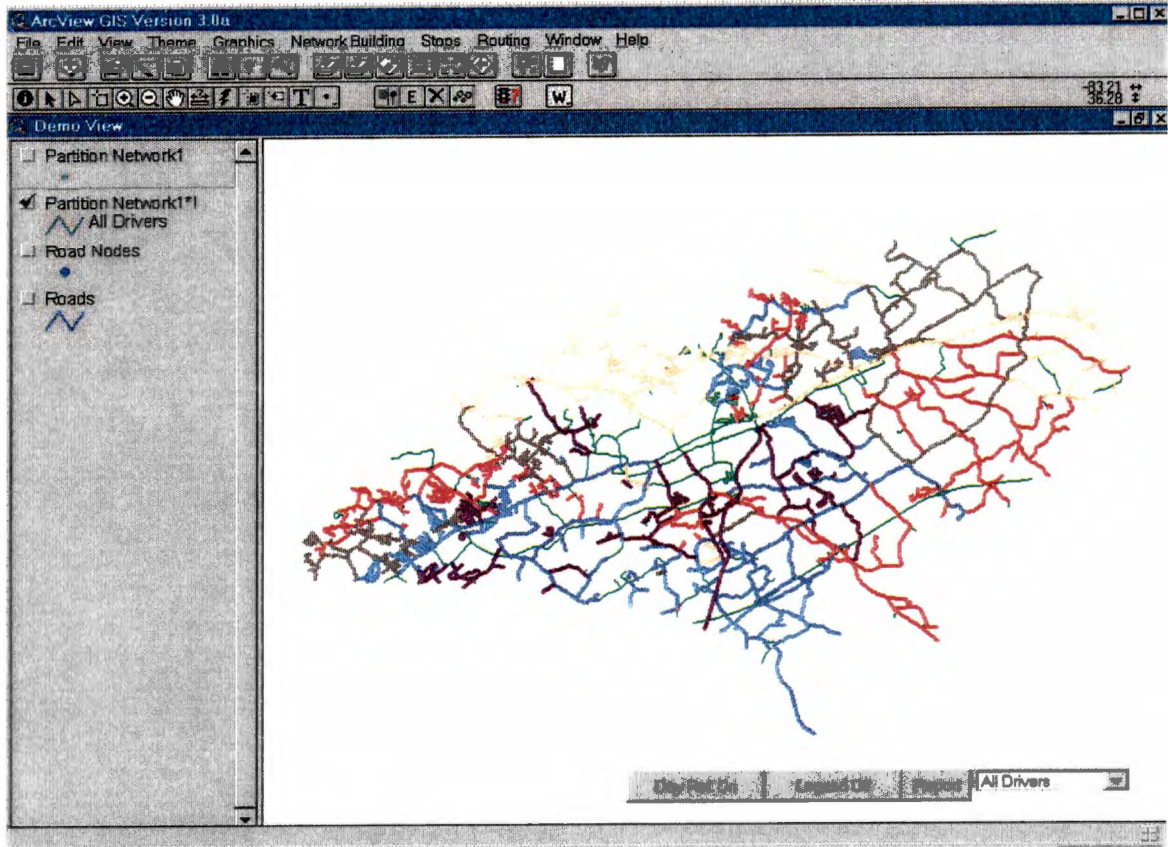


Figure 4.14 A Driver-Weekday Partition Interface with Subnetworks

4.3.2 Data Attributes for a Partition Network and its Subnetworks

It was discussed previously that a partition network is inherited from a base road network. In a partition network attribute table, two new data items are added to those from the base road network. They are "Driver" and "Day", which are designed to store a tabular expression of the latest partition network. As a matter of fact, a partition network is a template for generating subnetworks. Shape lines are graphic objects associated with no attribute tables but with object tags. These objects are used to carry data attributes of a base network to the attribute table of a partition network. Through the partition network, subnetworks are generated. Since a base road network is partitioned for driver partition for the first time and for driver-day combination partition for the second time, two kinds of objects and their object attributes needed to be addressed.

1. Object Tag Format for Driver Partition

<i>driver</i>	<i>DC</i>	<i>tpID</i>	<i>featureID</i>	<i>demand</i>	<i>new</i>
---------------	-----------	-------------	------------------	---------------	------------

- "driver" is a flag for recognizing the object for driver partition. It refers to driver partition and its interface.
- "DC" is an abbreviation of "Driver Code", which is an ID to identify drivers. For the Hamblen County SWRoute, driver codes for four drivers are "272", "275", "277", and "279" respectively.
- "tpID" is from the attribute table of a partition network link file and shown as a tabular item. It is the unique ID of topological relationships among line shapes in the partition network and the partition node shape file uses tpID as its ArcID¹⁴.

¹⁴ A field name in a node coverage, which is topologically cleaned in ArcInfo

- “featureID” is a hidden field in the attribute table of a partition network. It is accessed when spatial shapes are retrieved from the partition network through a bitmap. The index number of the bitmap is equivalent to “featureID”. The usefulness of “featureID” is for graphic shapes to directly fetch spatial objects and their attributes from their attribute table.
- “demand” is a field in the attribute table of a partition network. It represents an approximate amount of solid-waste collected per crew on the road¹⁵ per collection week.
- “new” is optional. It is a flag for objects created by tools.

2. Object Tag Format for Drivers-Weekday Partition

<i>day</i>	<i>DC</i>	<i>DW</i>	<i>featureID</i>	<i>demand</i>	<i>new</i>
------------	-----------	-----------	------------------	---------------	------------

- “day” is a flag for recognizing the object for drivers-weekday partition.
- “DC” is an abbreviation of “Driver Code”, which is an ID to identify drivers. Driver codes for the Hamblen County SWRoute are “272”, “275”, “277”, and “279”.
- “DW” is an abbreviation of “Driver-Weekday”, which represents weekdays when drivers work. DA for the Hamblen County SWRoute can be “Monday”, “Tuesday”, “Wednesday”, “Thursday”, and “Friday”.
- “tpID”, “featureID”, and “demand” all have the same meaning as their counterpart in driver partition tag format. They are reused for driver-day partitioning.
- “new” is optional. It is a flag for objects created by tools.

¹⁵ In a partition network, a road is represented by a link with two end nodes.

Interface functions for generating subnetworks are listed in Appendix C.

4.4 SWRoute Seed Node Set

4.4.1 Creating a Seed Node Set for the Hamblen County SWRoute

It is assumed that all seed nodes are located on the nodes of a SWRoute partition node coverage with unique ID number associated with the links in the SWRoute partition node coverage. The Hamblen County SWRoute seed node set is a collection of garages, landfills, and depots. The garage is a location where the Hamblen County SWRoute drivers start and end their daily tasks. The landfill is where SWRoute drivers dump trashes. The depot is a node from and to which drivers traverse links within driver-day subnetworks. A garage, a depot, and a landfill are required when a driver-day subnetwork is generated. In Hamblen County, there is only one garage and one landfill. The following section presents procedures of generating the seed node set for the Hamblen County by using tools available from SWRoute.

Step 1. Build a stop workspace for a seed node set

As shown in Figure 4.15, a workspace can be built from scratch by “Create A Stop Workspace” or from existing workspace “Open A Stop Workspace”. A stop workspace is represented by a seed node set. A stop workspace can be removed by the “Delete a Stop Workspace” menu option. There must be at least one seed node set existing in the SWRoute interface before the user proceeds to the next step. If more than one stop set is opened, the latest built one is considered as the active stop workspace.

Step 2. Populate a stop workspace with a seed node set

As shown in Figure 4.16, four menu items and their corresponding tool buttons exist for building a seed node set. Among them, “Add Garage/Landfill/Depots” is for creating a new seed at a location of a partition network and a dialog box is brought up for collecting information about the seed node. In Figure 4.17, “Unique Node ID”, “Stop Name”, and “Stop Address” are all read-only items. “Stop Type” can be chosen as either a garage, a landfill, or a depot. For a depot node, extra dialog items are available about the driver code and weekday, which are necessary for identifying a specific depot. After finishing adding a seed node, a new node appears on the map and its associated record is added to the seed node attribute table. An existing node can be edited with “Edit Stop Attributes” tool(Figure 4.18). All edited attributes are kept in the seed node attribute table. The “Delete Stops” tool is used for removing a node from the map and the seed node attribute table. “Move Stops” is a tool for moving a selected stop from one node location to another existing node location.



Figure 4.15 A Menu for Building SWRoute Seed Node Sets

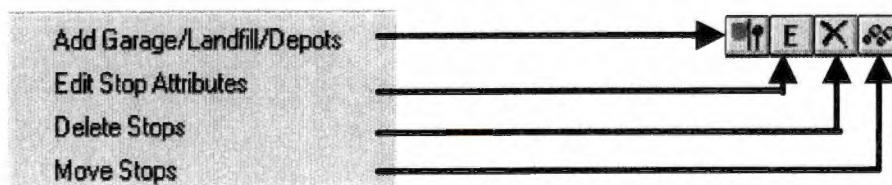


Figure 4.16 Menus and Tools for Generating SWRoute Seed Node Sets

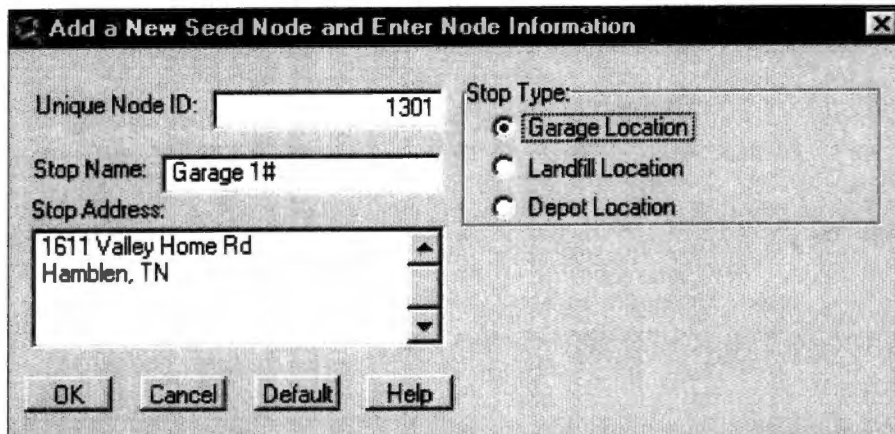


Figure 4.17 A Dialog Box Used for Adding a Garage

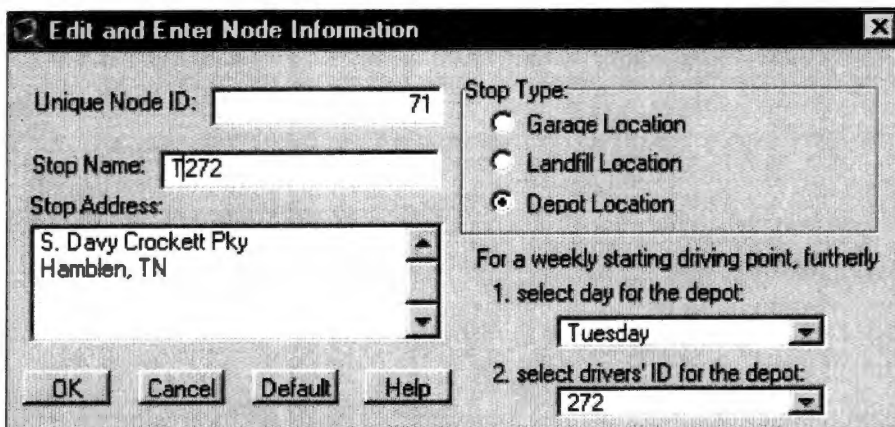


Figure 4.18 A Dialog Box Used for Editing a Depot

4.4.2 Data Attributes for a SWRoute Seed Node Set

The database for seed node attributes consists of the following fields.

<i>tpID</i>	<i>userID</i>	<i>name</i>	<i>Address</i>	<i>Type</i>	<i>driver</i>	<i>weekday</i>
-------------	---------------	-------------	----------------	-------------	---------------	----------------

1. “tpID” is a topological node ID for a partition node network and a partition link network.
2. “userID” is another unique ID incremented by the number of added nodes.
3. “name”, “address”, and “type” are information about a stop name, a stop address, and a stop type, which is a choice of garages, landfills, and depots.
4. “driver” and “weekday” are flags for recognizing the object for driver-day partition.

4.5 SWRoute Routes and Routing Reports

4.5.1 Generating SWRoute Routes and Reporting Routes

After subnetworks and a seed node set are created, SWRoute routes are generated. It is in this step that data from the GIS are passed to the algorithms discussed in Chapter 3. Report functions are designed to display route information in text and graphic format. Figure 4.19 shows the menu items and a report tool for routes. Procedures for generating and reporting routes are addressed in following steps.

Step 1: Build New Routes

Choosing “Build New Routes” will cause a route collection dialog box to be displayed. In Figure 4.20, a driver, his vehicle type, and his working weekdays can be selected. Since time scheduling is beyond the scope of the thesis, the time configuration is ignored and the default values are kept.

SWRoute has functions showing route generating messages (Figure 4.21). For one



Figure 4.19 Menus and a Tool for Generating and Reporting SWRoute

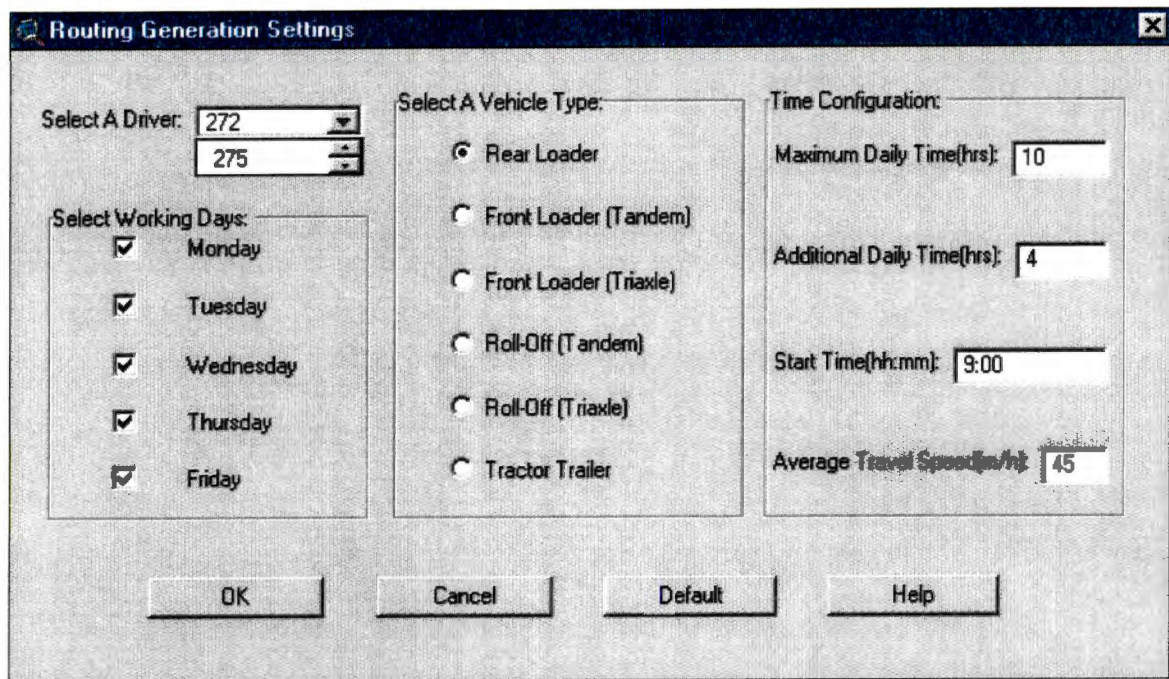


Figure 4.20 A Route Configuration Dialog Box

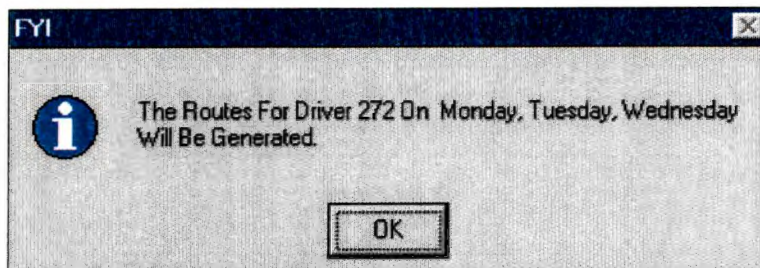


Figure 4.21 A Message Box for Checking Seed Nodes and their Associated Routes

driver on one weekday, one route based on a depot and two shortest paths from a garage to a depot and from a depot to a landfill are generated.

Step 2. Show/Hide New Routes

A dialog box (Figure 4.22) is used for showing or hiding new routes. Items listed in the dialog box are all routes newly generated and available for showing. If an item is selected, a route with that item name is shown or refreshed on the map. Not selected means the route will be removed from the map. For example, all three routes in Figure 4.22 are selected and they are shown as new routes on the map. If the three routes have already been in the view, the old ones are removed and new ones with the same names are added. They might be different since they are generated for a new route for a same driver on a same weekday. Figure 4.23 shows the routes generated and shortest paths between seed nodes. Links with thicker line weight are links to be traversed. Arrows on the links indicate traversing directions.

Step 3. Report Routes

The menu item "Report Routes" and its associated tool are used to report which route a link relates to, the workload of the route, the nodes traversed by the route, and the route traversing direction. Report results are shown in a notepad (Figure 4.24) and graphics on the map (Figure 4.25).

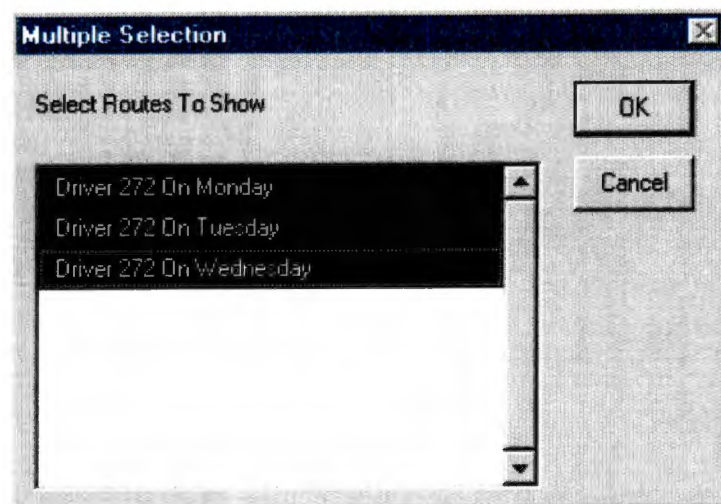


Figure 4.22 A Dialog Box for Showing/Hiding Routes

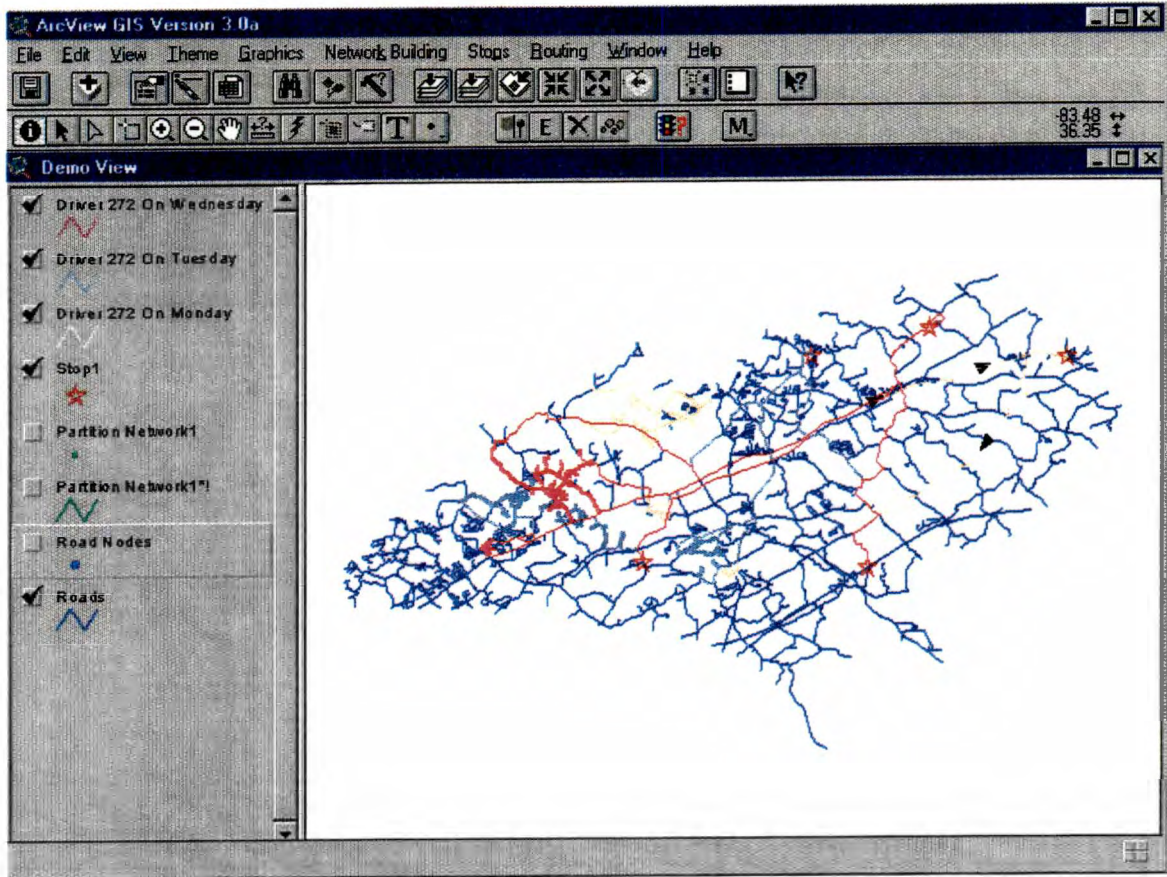


Figure 4.23 Routes for Driver 272 on Monday, Tuesday, and Wednesday

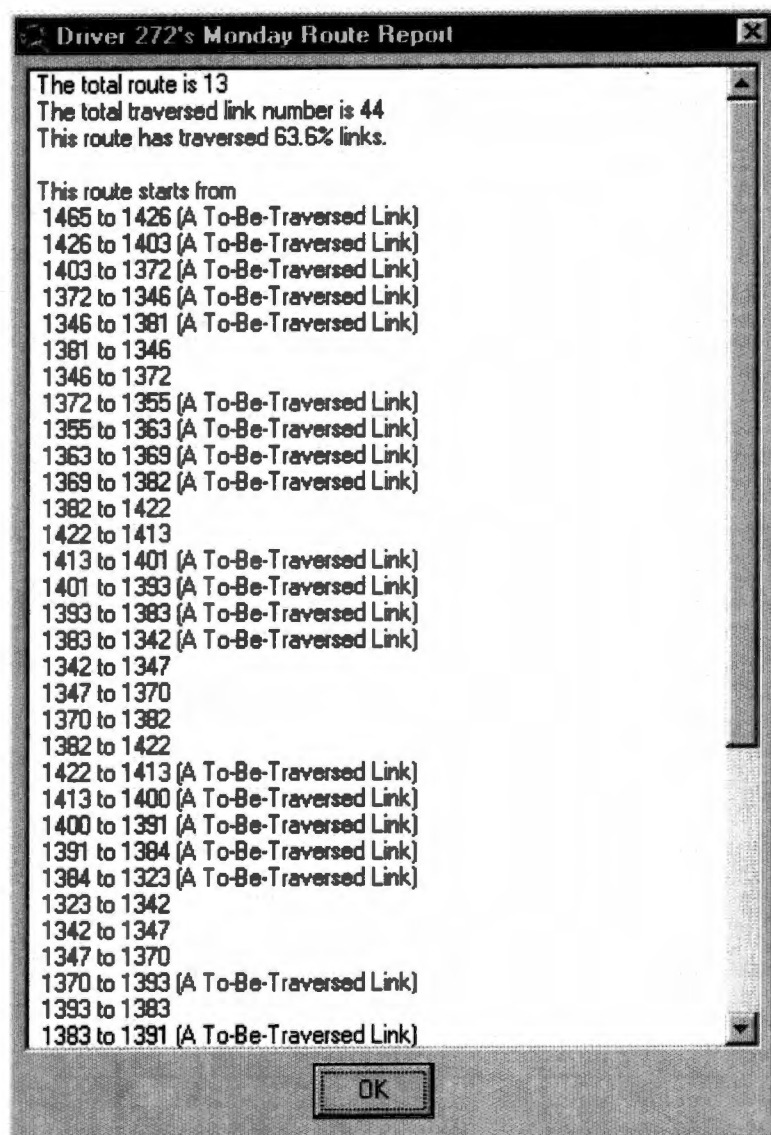


Figure 4.24 A Notepad Route Report for Driver 272 on Wednesday

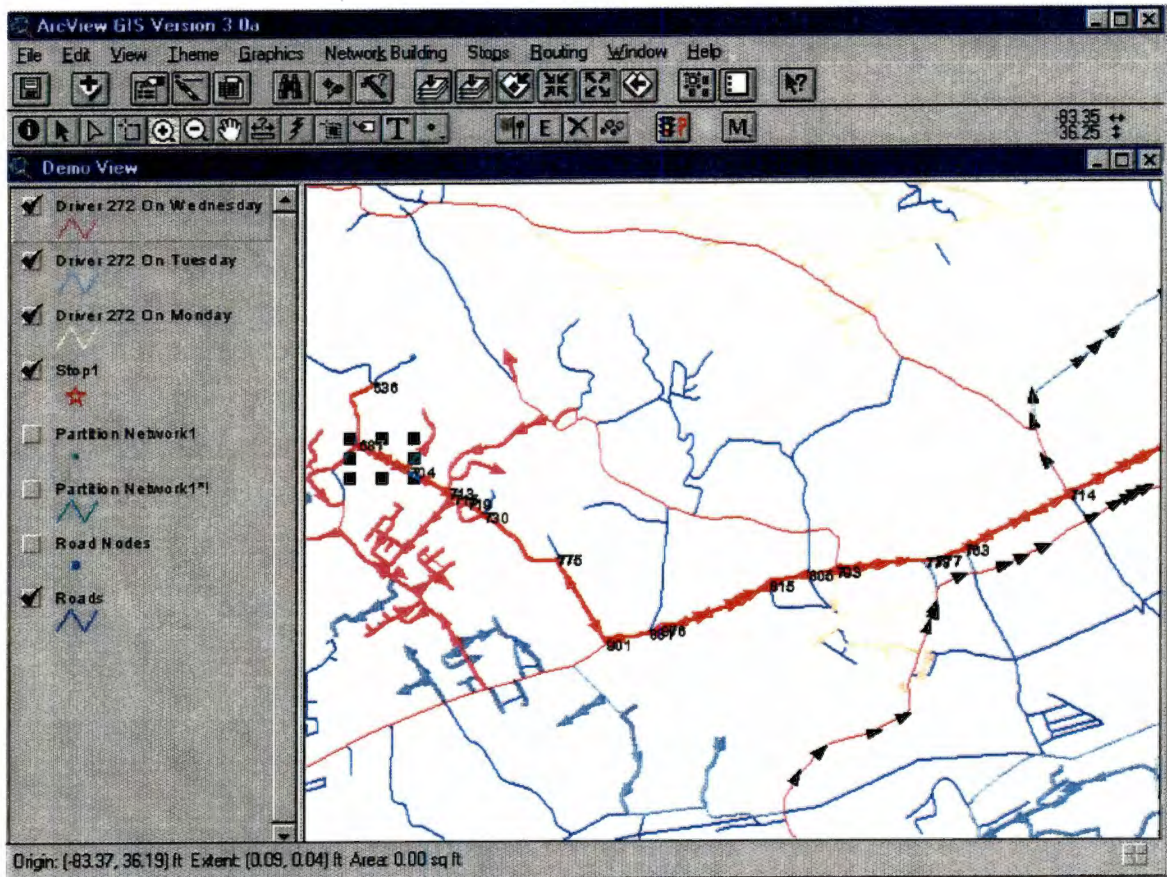


Figure 4.25 A Graphic Route Report for Driver 272 on Wednesday

4.5.2 Data Attributes for a Route Object

A route object consists of graphic shapes with object tags. Its object tag carries information about the route, which is derived from route attribute tables. The object tag is composed of following items.

<i>driver-dayID</i>	<i>tpID</i>	<i>FeatureID</i>	<i>demand</i>	<i>routeID</i>
<i>OrderID</i>	<i>routeFnode</i>	<i>RouteTnode</i>	<i>traverseBool</i>	<i>distance</i>

1. “driver-dayID” is a combination of driver code and weekday and they are linked by “-”. dayID is an index number for weekdays. They are 1, 2, 3, 4, 5 for Monday, Tuesday, Wednesday, Thursday, and Friday. Therefore, driver-dayID can be written as 272-3, which means driver 272 on Monday. “driver-dayID” is useful for identifying a series of route file names. All files associated with a driver-dayID are about that driver on a weekday. For example, r272-3.txt is an ASCII file of generated routes for driver 272 on Wednesday and sht2722-3.txt is also an ASCII file generated for shortest paths for the same driver on the same day.
2. “tpID” and “featureID” are two unique ID for representing graphic shapes’ tabular attribute and their graphic attributes.
3. “demand” is the amount of trash collected by the driver on a weekday.
4. “routeID” is an index number for routes. It is generated from writing an ASCII route and shortest path output files to a route shape file.
5. “orderID” is another index number for links belonging to one route. It is also generated from writing an ASCII route and shortest path output files to a route shape file. It is significant for identifying the order links to be traversed for a route.

6. "routeFnode" and "routeTnode" are Ids of from nodes and to nodes.
7. "traverseBool" is a Boolean variable and represented by -30 and -10. -30 means yes and a link is to be traversed. -10 means no and a link is not to be traversed.
8. "distance" is an alternative of link length.

Interface functions for route generating and reporting are listed in Appendix C.

4.6 Concluding Remarks

In the Hamblen County SWRoute, a base road network is partitioned into 20 subnetworks because there are four crews working on five weekdays each week. The algorithms presented in chapter 3 are used for generating routes for these 20 subnetworks respectively. This chapter presents principles for designing SWRoute GIS interface and its toolboxes. These toolboxes have functions for 1) partitioning a base road network into subnetworks, 2) creating seed node set, and 3) generating and reporting routes. The first two toolboxes collect user inputs about SWRoute partition and stops. The third integrate the route generating algorithms and shows the result in the GIS interface.

CHAPTER 5

Discussion and Recommendations

5.1 Summary of research results

In the light of the results presented in the previous chapters, this chapter answers the research questions that were posed in chapter 1. The questions raised in Chapter 1 were:

Can a prototype GIS-assisted system be built with decision support functions, which can be used to develop, apply, and evaluate a wide range of solid-waste collection route systems in rural U.S. counties?

Its related questions were:

Question 1: What are the advantages and the disadvantages of a methodology of linking GIS and SDSS for designing solid-waste collection routes?

Question 2: Can we classify the algorithms currently used for solving arc routing problems into useful sets of approaches and processes?

Question 3: What kinds of heuristic algorithms present a reasonable solution to the solid-waste collection routes?

Question 4: Can we build tools with general purposes to match those algorithms?

The linkage between GIS and SDSS for implementing SWRoute is not loosely coupled. There is a full integration of GIS functionality for manipulation of input, results, and formulation of the models required by SWRoute.

In general, the advantages of integrating GIS and SDSS are listed below:

1. The GIS and SDSS integration allows for a compact expression of the problem by hiding SWRoute implementation details. The Hamblen County SWRoute interface was designed within GIS software for formatting, solving, and studying solutions to solid-waste routing modeling problems. Users have full access to the information concerning the solution without having to know the details of the mathematical problem that is solved.
2. In a GIS-based SDSS, there are convenient tools designed for conversion between GIS and modeling system and between individual submodels. The development of the GIS-based SDSS begins with a classification of approaches for a specific type of problem into a limited number of categories, and subdivides each approach into separate generic tasks. For the Hamblen county SWRoute, this yields a framework of three submodels including partitioning subnetworks, adding a seed node set, and running arc routings for subnetworks. The GIS-based SDSS has functions for integrating data communication and conversion between these three submodels.

The disadvantages of the integration of GIS and SDSS for SWRoute are:

1. The current generation of commercial GIS software does not support GIS-based SDSS for SWRoute and its modeling implementation – arc routing problems. Popular GIS companies, such as ESRI and Intergraph, are developing network modules to solve vehicle routing problems and other location-allocation problems. However, no modules are currently available for arc routing problems by these vendors. This fact showed that a GIS-based SDSS SWRoute had to be built from scratch, which increases the difficulties and lengthens the time of designing an arc routing system.

2. Investment in developing tools and functions is high. Besides the difficulties mentioned above, much effort is needed to find an approximate algorithm, to implement arc routing models, and to design tools and functions in order to combine models with a GIS interface to show the result from the algorithm. Although GIS software provides some basic tools to use, in their native state they do not meet the needs of arc routing problems.

The next step in the development of a GIS-based SDSS for SWRoute is the implementation of the generic concepts derived from the analysis of specific individual models into functions and submodels. The submodules are capable of performing the generic tasks from which the total model can be constructed. The main module for SWRoute is to implement algorithms for arc routing problems. As discussed in Chapter three, the augment-merge algorithm is developed for computing SWRoute candidate solutions and Pearn's algorithm (1988) for computing its corresponding lower bounds. Other supporting algorithms are needed for shortest path searching and the nonbipartite minimum cost perfect matching problem. In this thesis, these algorithms are all regarded as modules, and are implemented as DLLs called from the GIS.

To solve the CARP, this thesis applied the augment-merge algorithm. This algorithm is applicable to the circumstances of the Hamblen County. First, it is an algorithm used for a transportation network with sparsely distributed arcs with non-zero demand for solid-waste collection, which describes the Hamblen County situation. Second, the augment-merge algorithm is based on the shortest path algorithms, which are well developed. Thirdly, by computing lower bounds of available routes, we can determine the quality of the solutions from the augment-merge algorithm.

As a heuristic algorithm, the augment-merge algorithm can not guarantee optimal solutions for SWRoute. However, it provides an opportunity to develop good solutions which can be evaluated by local experts on a GIS. The GIS interface developed for SWRoute contains tools to implement all the functionality for creating a transportation network, constructing SWRoute, and evaluate its solutions. This raises questions about the robustness of the tools in the hands of non-technical persons. The tools are helpful in designing and structuring SWRoute, and shield the user from the details of the implementation of the algorithm and model. This enables the user to focus on the geographic nature of the problem and its solution. Since SWRoute is integrated in a GIS environment, all common GIS tools for data extraction and mapping are available. The SWRoute tools are integrated with the standard GIS tools in a seamless way.

5.2 Recommendations for Future Research

According to Arjang A. Assad, et al. (1988), "the major advance in vehicle routing has been to capture enough characteristics of the real-world distribution environment to enable the solution procedures to obtain a useful answer, without thereby precluding their computational tractability". They further claim that this desirable state of affairs has resulted from a combination of "careful modeling, the design of clever heuristics, and an interactive user interface". This thesis provides for a set of tools to make the solution of arc routing problem in a GIS-based SDSS as easy and as flexible as possible. SWRoute allows for the formulation of arc routing problems suitable for different regional environments: The Hamblen County SWRoute illustrates how SWRoute can be tailored to a real world problem. In the future, more research needs to be

done to improve SWRoute to achieve goals regarding the characteristics of the routing environment constraints on route design, alternative routing algorithms, and implementations to the SWRoute user interface.

1. Modeling Extensions

Future research should consider possible complexities of arc routing problems with respect to solid-waste generating characteristics and operational routes.

- Solid-waste can be distinguished by the uncertainty in the amount and types generated from different counties. In some cases, the amount actually collected may vary from predicted levels. SWRoute classifies all solid-waste as residential and uses one fixed generated rate for the entire county during all seasons of a year. This simplification might not apply.
- SWRoute takes the start and end locations of a route as the same depot. However, the problem becomes more complicated when certain vehicles have full loads and can exit from a node which is not a starting point.
- SWRoute ignores the scheduling of drivers' routes (service time window) and the additional constraints that reflect driver safety regulations. In service activities such as solid-waste collection routing problems, the timing of services generally assumes a larger importance. This fact requires combining routing and scheduling to seek for good spatial and temporal configurations for routes.

2. Algorithm Extensions

The current objective of SWRoute is to minimize distance-related costs over the fleet of vehicles or just the number of vehicles used. The objective function can be expanded to ensure that balanced routes are generated and penalize large imbalances.

Similar consideration applies to guaranteeing a certain minimum number of hours of work each day. Moreover, additional constraints might be added to reflect the scheduling of crews.

In the vehicle routing research literature, the augment-merge algorithm belongs to a category of heuristic problems termed as “sequential heuristics”, which means the problem is decomposed into a sequence of subproblems that are sequentially solved with different algorithms. SWRoute uses a “cluster first-route second” approach to obtain acceptable solutions and with the aid of nonbipartite minimum cost matching problems, a lower bound is computed to evaluate the approximation of the solution. Using lower bounds indicate how good or how bad the solution is, but it can not tell how to achieve a better solution.

3. GIS Interface Extensions

SWRoute is built on a topological road network. Although ArcView provides some tools for editing the tabular attributes of the network, no special tools are designed for SWRoute so that it is impossible for a person who does not know ArcView to edit SWRoute attributes.

The current interface is used for preparing input data (constructing “what-if?” scenarios) and displaying model outputs. There is no ability to allow user intervention with the algorithmic operations, such as swapping routes or sections of routes between driver-day partitions.

The goal of future SWRoute is to improve the ability of planners to design solid-waste routes. This perspective requires that we learn from the experiences of the users of SWRoute in order to create a more useful spatial decision support system. Research

needs to be conducted on how to improve SWRoute. Individuals involved in routing solid-waste vehicles are the best judges of the need for new tools, model extensions, and user interface improvements.

BIBLIOGRAPHY

Bibliography

- Adam, Nabil R. and Aryya Gangopadhyay (1997) *Database Issues in Geographic Information Systems*, Kluwer Academic Publishers, Boston.
- Assad, A., W. L. Pearn, and B. L. Golden (1978) "The Capacitated Chinese Postman Problem: Lower Bounds and Solvable Cases", *American Journal of Mathematics and Management Science*, 7, 63-88.
- Belenguer, E. Benavent (1991) "Polyhedral Results on the Capacitated Arc Routing Problems", *Working Paper*, 55 and 57, Department De Extadistrictica e Investigacion Operativa. Universidad de Valencia, Spain.
- Beltrami, E. and L. Bodin (1974) "Networks and Vehicle Routing for Municipal Waste Collection", *Networks*, 4, 65-94.
- Benavent, E., V. Campos, A. Corberan, and E. Mota. (1992) "The Capacitated Arc Routing Problem: Lower Bounds", *Networks*, 22, 669-690.
- Berge (1957) "Two Theorems in Graph Theory", *Proceedings of the National Academy of Sciences USA*, 43, 842-844.
- Bodin, L. and Samuel J. Kursh (1978) "A Computer-Assisted System for the Routing and Scheduling of Street Sweepers", *Operations Research*, 26(4), 525-537.
- Bodin, L. and Samuel J. Kursh (1979) "A Detailed Description of a Computer System for the Routing and Scheduling of Street Sweepers", *Comput. & Ops Res.*, 6, 181-198.
- Bodin, L. and B. Golden (1981) "Classification in Vehicle Routing and Scheduling", *Networks*, 11(2), 97-108.
- Bodin, L. and Laurence Levy (1991) "The Arc Partitioning Problem", *European Journal of Operational Research*, 54, 391-401.
- Bumpus, Lewis D. *Solid-waste: Transportation and Other Costs*, County Technical Assistance Service, The University of Tennessee, Knoxville.
- Busch, K. (1991) *Vehicle Routing on Acyclic Networks*, Ph.D. Dissertation, The Johns Hopkins University, Baltimore, MD.
- Christofides, N. (1979) "The Optimal Traversal of a Graph", *Omega*, 1, 719-732.

Christofides, N., A. Mingozzi, and P. Toth (1981) "Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations", *Math. Prog.*, **20(3)**, 255-282.

Clark, Roberts and John C. H. Lee. Jr. (1976) "Systems Planning for Solid-waste Collection", *Comput. & Ops Res.*, **3**, 157-173.

County Technical Assistance Service (1992) *Hamblen County Solid-waste Needs Assesment*, East Tennessee Development District, The University of Tennessee, Knoxville.

Corry, Chris (1994) *Killer Borland C++ 4*, QUE Corporation, NH.

Crossland, M.D., B.E. Wynne, and W.C. Perkins (1995) "Spatial Decision Support Systems: An Overview of Technology and a Test of Efficiency", *Decision Support System*, **14**, 219-235.

Dallaire, Gene (1996) "How Cities Are Using GIS for Route Optimization", *MSW Management*, May/June, 74-79.

Daniel, Larry (1992) "SDSS for Location Planning, or The Seat of the Pants is Out", *GeoInfo Systems*.

"Decision Support System, Environmental Models, Visualisation Systems and GIS", <http://bamboo.mluri.sari.ac.uk/~jo/litrev/chap5.html>

Edmonds, J. (1965) "Path, Trees, and Flowers", *Canadian Journal of Mathematics*, **17**, 449-467.

Edmonds, J. (1965) "Matching and a Polyhedron with 0-1 vertices", *J. Res. Nat. Bur. Standards Sect. B.*, **69**, 125-130.

Eiselt, H. A., Michel Gendreau, and Gilbert Laporte (1995a) "Arc Routing Problems, Part I: The Chinese Postman Problem", *Operations Research*, **43(2)**, 231-242.

Eiselt, H. A., Michel Gendreau, and Gilbert Laporte (1995b) "Arc Routing Problems, Part II: The Rural Postman Problem", *Operations Research*, **43(3)**, 399-414.

Enache, Mircea (1994) "Integrating GIS with DSS: A Research Agenda", *URISA*, 154-166.

Gabow, H. N. (1973) *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs*, Ph.D. dissertation, Dept. Electrical Engineering, Stanford Univ., Stanford, CA.

Gabow, H. N. and R. E. Tarjan (1983) "A Linear-time Algorithm for a Special Case of Disjoint Set Union.", *Proc. Fifteenth Annual ACM Symposium on Theory of Computing*, 246-251.

Galil, Z., S. Micali, and H. Gabow (1983) "Maximal Weighted Matching on General Graphs", *Proc. 23rd Annual ACM Symposium on Foundation of Computer Science*, 255-261.

Golden, B. L. and A. A. Assad (1988) *Vehicle Routing: Methods and Studies*, Elsevier Science Publishers B. V., The Netherlands.

Golden, B., T. Maganti, and H. Nguyen (1977) "Implementing Vehicle Routing Algorithms", *Networks*, 7, 113-148.

Golden, B. and R. Wong (1981) "Capacitated Arc Routing Problems", *Networks*, 11, 305-318.

Golden, J. DeArmon and E. K. Baker (1983) "Computational Experiments with Algorithms for a Class of Routing Problems", *Computers Operations Research*, 10, 47-69.

Haagsma, Ijsbrand G. and Remco D. Johanns "Decision Support Systems, an Integrated and Distributed Approach", <http://cci.ct.tudelft.nl/Remco/SanFrancisco.htm>

Hamblen County (1994) *Hamblen County Regional Solid-waste Plan*.

Ji, Wei and James Johnson (1994) "A GIS-Based Decision Support System for Wetland Permit Analysis", *GIS/LIS*, 471-476.

Keenan, P. and M. Naughton (1995) "Arc Routing for Rural Irish Networks", In J. Dolezal & J. Fildler (Eds.), *System Modelling and Optimization*, Chapman Hall, London.

Keenan, P., Harold C. Harrison, and Andrew J. Deegan (1996) "A Decision Support System for Arc Routing", Working Paper, University of College Dublin, Ireland.

Koenig, Andrew (1989) *C Traps and Pitfalls*, AT&T Bell Laboratories, Addison-Wesley Publishing Company.

Lawler, L. (1976) *Combinatorial Optimization: Networks and Matriods*, Holt, Rinehart and Winston, New York.

Lenstra, J.K. and A.H.G. Rinooy Kan (1976) "On General Routing Problems", *Networks*, 6, 273-280.

Magnanti, T. (1981) "Combinational Optimization and Vehicle Fleeting Planning: Perspective and Prospects", *Networks*, 11(2), 279-213.

McBride, Richard (1982) "Controlling Left and U-Turns in the Routing of Refuse Collection Vehicles", *Comput. & Ops Res.*, **2**, 145-152.

Moon, George and Mark Ashworth "Capacities Needed in Spatial Decision Support Systems".

Mullaseril, Paul Abraham (1997) *Capacitated Rural Postman Problem with Time Windows and Split Delivery*, Ph.D dissertation, The University of Arizona.

Murty, Katta G. *Network Programming*, Prentice Hall, Englewood Cliffs, New Jersey

Nagel, Stuart S. (1992) *Applications of Decision-Aiding Software*, St. Martin's Press, N.Y.

Negreiros, M. *O Problem de Palnejamento e Percuso de Veículos na Coleta do Lixo Urbano Domiciliar*, M.Sc. Dissertation, Sistemas/COPPE, Federal University of Rio de Janeiro, Brazil.

Pearn, W. (1988) "New Lower Bounds for the Capacitated Arc Routing Problem", *Networks*, **18**, 181-191.

Pearn, W. (1989) "Approximate Solutions for the Capacitated Arc Routing Problems", *Computers Operations Research*, **16**, 589-600.

Pearn, W. (1991) "Augment-Insert Algorithms for the Capacitated Arc Routing Problem", *Computers Operations Research*, **18(2)**, 189-198.

Ralston, Bruce (1994) "Object-Oriented Spatial Analysis", in *Spatial Analysis and GIS*, S. Fotheringham and V. Rogerson (Eds.), Taylor & Francis Ltd., London, 165-185.

Ralston, Bruce, George Tharakan, and Cheng Liu "A Spatial Support System for Transportation Policy Analysis", *Journal of Transportation Geography*, **2(2)**, 101-110.

Ray, Julian J. (1990) *The Flow-Resource Facility Location Problem*, Ph.D dissertation, The University of Tennessee, Knoxville.

Russell, R. (1977) "An Effective Heuristic for the M-Tour Travelling Saleman Problem with Some Side Conditions", *Operation Research*, **25**, 517-524.

Smelcer, John B. and Erran Carmel (1994) *Do Geographic Information Systems Improve Decision Making? An Experiment Comparing Maps and Tables*, <http://lattanze.loyola.edu/lattanze/research/wp1094.023.htm>

Sprague, Ralph H. Jr. and Hugh J. Watson (1989) *Decision Support Systems: Putting Theory into Practice*, Edited, Prentice Hall, Englewood Cliffs, New Jersey.

Sticker, R. (1970) *Public Sector Vehicle Routing: The Chinese Postman Problem*, M.Sc. Dissertation, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass.

Swan, Tom *Mastering Borland C++*, SAMS, A Division of Prentice Computer Publishing, Indiana.

Tarjan, Robert Endre (1983) *Data Structures and Network Algorithms*, Bell Laboratories, Murray Hill, New Jersey.

Worboys, Michael F. (1995) "GIS: A Computing Perspective", Taylor & Francis Ltd, London.

APPENDICES

APPENDIX A – A CARP MATHEMATICAL FORMULATION

Two integer linear programming formulations have been proposed for the CARP on the basis of directed and undirected networks, respectively. Though the thesis defines a directed network for SWRoute with all arcs with two-way links, it is necessary to clarify the difference between the mathematical formulations for directed and undirected networks in order to understand its undirected network nature.

In the directed CARP formulation (Golden and Wang, 1981), a mathematical programming formulation for the CARP is given as follows:

$$\text{Min } \sum_{i=1}^n \sum_{j=1}^n \sum_{p=1}^k c_{ij} x_{ij}^p \quad (1)$$

subject to:

$$\sum_{k=1}^n x_{ki}^p - \sum_{k=1}^n x_{ik}^p = 0 \quad \forall i = 1, \dots, n \quad (2)$$

$$\forall p = 1, \dots, K$$

$$\sum_{p=1}^k (l_{ij}^p + l_{ji}^p) = \left\lceil \frac{q_{ij}}{W} \right\rceil \quad \forall (i, j) \in A \quad (3)$$

$$x_{ij}^p \geq l_{ij}^p \quad \forall (i, j) \in A \quad (4)$$

$$\forall p = 1, \dots, K$$

$$\sum_{i=1}^n \sum_{j=1}^n l_{ij}^p q_{ij} \leq W \quad \forall p = 1, \dots, K \quad (5)$$

$$\left. \begin{aligned}
\sum_{i \in \tilde{Q}} \sum_{j \in \tilde{Q}} x_{ij}^p - n^2 y_{1\tilde{q}}^p &\leq |\tilde{Q}| - 1 & (6) \\
\sum_{i \in \tilde{Q}} \sum_{j \in \tilde{Q}} x_{ij}^p + y_{2\tilde{q}}^p &\geq 1 & (7) \\
y_{1\tilde{q}}^p + y_{2\tilde{q}}^p &\leq 1 & (8) \\
x_{ij}^p, l_{ij}^p, y_{1\tilde{q}}^p, y_{2\tilde{q}}^p &\in \{0,1\} & (9)
\end{aligned} \right\} \begin{aligned}
&\forall p=1, \dots, K \\
&\forall \tilde{q}=1, \dots, 2^{n-1} - 1 \\
&\forall \tilde{Q} \subseteq N \setminus \{0,1\}, \tilde{Q} \neq \phi
\end{aligned}$$

where:

n = the number of nodes.

K = the number of available vehicles.

q_{ij} = the demand on arc (i, j) .

W = the vehicle capacity ($W \geq \max q_{ij}$).

c_{ij} = the length of arc (i, j) .

$x_{ij}^p = 1$, if arc (i, j) is traversed by vehicle p , 0 otherwise.

$l_{ij}^p = 1$ if a vehicle traverses and services arc (i, j) , 0 otherwise.

$\lceil z \rceil$ = the smallest integer greater than or equal to z .

$y_{1\tilde{q}}^p$ and $y_{2\tilde{q}}^p$ are two variables to eliminate illegal subtours¹⁶; each combination index \tilde{q} corresponds to the set \tilde{Q} .

The objective function (1) seeks to minimize total distance traveled. Constraints (2) are conservation of flow constraints, which guarantee there are equal number of vehicles coming into and out the node i . Constraints (3) state that each arc with positive demand is serviced exactly once because there exists the assumption $W \geq \max q_{ij}$.

Constraints (4) guarantee that arc (i, j) must be serviced by vehicle p if the vehicle traverses arc (i, j) . Constraints (5) make sure that vehicle capacity W is not violated. Constraints (6), (7), and (8) are subtour breaking constraints. Constraints (6) ensure that the solution must contain no cycle using the nodes $2, 3, \dots, n$ (i.e., contain any subtours on these nodes). Constraints (7) make sure there exists a linkage between a subtour and its complement subtour if $y_{1\bar{q}}^p = 0$. Constraints (8) are for $y_{1\bar{q}}^p$ and $y_{2\bar{q}}^p$, which are variables for balancing constraints (6) and constraints (7). In other words, if $y_{1\bar{q}}^p = 1$ and $y_{2\bar{q}}^p = 0$, constraints (7) is not binding and constraints (6) independently address there are no illegal subtours. Constraints (8) are integer constraints.

In the formulation proposed by Belenguer and Benavent for the undirected unicursal¹⁷ network, the mathematical formulation is expressed as follows:

$$\text{Min } \sum_{i=1}^n \sum_{j=1}^n \sum_{p=1}^k c_{ij} (x_{ij}^p + l_{ij}^p) \quad (1)$$

subject to:

$$\sum_{p=1}^K l_{ij}^p = 1 \quad \forall i, j = 1, \dots, n \quad (2)$$

$$\forall p = 1, \dots, K$$

$$\sum_{i=1}^n \sum_{i=1}^n l_{ij}^p q_{ij} \leq W \quad \forall p = 1, \dots, K \quad (3)$$

$$\sum_{i \in E(S)} \sum_{j \in E(S)} x_{ij}^p + \sum_{i \in E^+(S)} \sum_{j \in E^+(S)} l_{ij}^p \geq 2l_{hl}^k$$

¹⁶ A tour which passes through only a subset of the nodes in a network is called a subtour. An illegal subtour is a tour starting from a depot node but never coming back to the depot node.

¹⁷ A unicursal or Eulerian network refers to a connected graph where there exists a closed walk in G containing each arc exactly once and each vertex at least once.

$$\forall A(S) = \{(i, j): i \in S \text{ and } j \in V \setminus S \text{ or } j \in S \text{ and } i \in V \setminus S. \text{ where } S \subseteq N\}$$

$$A^+(S) = A(S) \cap \{(i, j) \in A; d_{ij} > 0\}$$

$$\forall p = 1, \dots, K$$

$$\forall h \in S, l \in S \text{ and } q_{hl} > 0 \quad (4)$$

$$\left. \begin{array}{l} \sum_{i \in E(S)} \sum_{j \in E(S)} x_{ij}^p + \sum_{i \in E^+(S)} \sum_{j \in E^+(S)} l_{ij}^p = 2z_k^s \\ z_k^s \geq 0 \text{ and integer} \end{array} \right\} \forall p = 1, \dots, k \quad (5)$$

$$x_{ij}^p \geq 0 \text{ and integer} \quad (6)$$

$$l_{ij}^p \in \{0,1\} \quad (7)$$

where:

x_{ij}^p is defined only as a deadheading arcs, representing the number of times arc (i, j) is traversed by vehicle p without being serviced by that vehicle.

S is a subset of node set N .

$A(S)$ is a set of deadheading arcs starting from one cycle and ending to different cycle.

$A^+(S)$ is a subset of $A(S)$ including arcs with positive demand.

The objective function (1) intends to minimize the total distance of deadheading and serviced arcs. Constraints (2) state that every arc with a positive demand is serviced exactly once by a vehicle. Constraints (3) are capacity constraints, making sure that the total demand for a route will not surpass the vehicle capacity. Constraints (4) play a role to clarify that once an arc (i, j) in $A(S)$ is serviced by a vehicle, the arc (i, j) is also traversed by the same vehicle, which forms a continuous route. Constraints (5) address a

unicursal network for the undirected case, which means that any node $A(S)$ must be connected to traversed and serviced arcs an even number of times for any vehicle.

Constraints (6) and constraints (7) are integer constraints for x_{ij}^p and l_{ij}^p .

APPENDIX B - A UNDIRECTED MATCHING NETWORK AND NONBIPARTITE MINIMUM COST PERFECT MATCHING PROBLEM (NMCMPM)

As defined in CARP, let $G(N, A, C)$ still be a given directed network. Let R be the set of arcs with nonzero demand. Let $D(R, i)$ be the number of arcs from R incident to node i ($i \in N$). Let $SPL(i, j)$ be the cost of the shortest path from i to j . Let Q be the total arc demands in the network G .

Let $G'(N', E', C')$ be a derived undirected network from G . Its node set N' consists of all odd-degree nodes in N and its arc set E' consists of arcs traversed through the shortest path between every pair of nodes in N' . C' is the cost matrix for the shortest path distance between any two nodes. Besides these, there are other terms to be introduced for better understanding matching algorithms in the setting of matching theory.

Nonbipartite Minimum Cost Perfect Matching Problem

A matching M of a graph G' is a subset of arcs $M \subset E'$ that contains at most one matching arc incident at node i , for each $i \in N'$. The perfect matching of the graph G' is a matching M that contains exactly one edge incident to the node $i \in N'$. The size $|M|$ of M is the number of edges it contains. If a perfect matching M exists in G' , $|N'|$ must be even, and $|M| = |N'| / 2$. The cost of M is the sum of its arc costs, which is defined as $\sum (c_{ij} : \text{over } (i, j) \in M)$. A nonbipartite network refers to a G' which can not be partitioned its node set into two subsets N_1 and N_2 so that for each edge (i, j) in E either

$i \in N_1$ and $j \in N_2$ or $i \in N_2$ and $j \in N_1$. The nonbipartite minimum cost perfect matching problem (NMCPMP) is that of finding a matching of perfect matching with minimum weight in a nonbipartite network. Augmentation and blossoms are two terms associated with solving NMCPMP and are defined as follows.

Alternating Path and Augmentation

According to Edmonds (1965), let M be a matching of a graph G' . An arc in M is a matching arc and every arc not in M is unmatched. A node is matched if it is incident to a matching arc and unmatched otherwise. An alternating path is a simple path whose arcs are alternatively matching and unmatched. The length of an alternating path is the number of arcs it contains. An alternating path is augmenting if both its ends are unmatched nodes. If M is an augmenting path then M is not of maximum size, its size becomes $|M| + 1$ by interchanging matching and unmatched arcs along the path. Moreover, all the matched nodes in M remain matched and two additional nodes from the both ends of the path are matched. A well-known theorem by Berge (1957) states that a matching M has a maximum matching if and only if G contains no augmenting path with respect to matching M .

As an example, consider the paths $P_1 = 2, (2, 5), 5, (5, 8), 8, (8, 9), 9$; $P_2 = 1, (1, 2), 2, (2, 5), 5$; $P_3 = 1, (1, 3), 3, (3, 4), 4, (4, 6), 6$ in the network in Figure B.1. All these are alternating paths with the matching arcs drawn as thick lines. Rematching the thick matching using P_1, P_2, P_3 leads to the matchings: $M_1 = \{(5, 8), (7, 10), (3, 4)\}$, $M_2 = \{(1, 2), (8, 9), (7, 10), (3, 4)\}$, $M_3 = \{(1, 3), (4, 6), (7, 10), (8, 9), (2, 5)\}$ respectively. It

can be verified that among these three alternating paths, only P_3 is an augmenting path because rematching using it increases the length of the matching from 4 to 5.

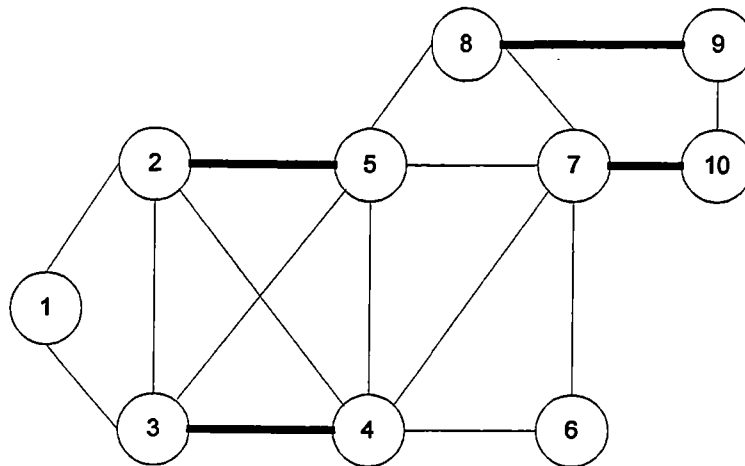


Figure B.1. A Matching Network with Thick Matching Edges

Flowers and Blossoms

One unique property of nonbipartite matching problem is the presence of certain subgraphs composed of particular types of paths and odd cycles, which add more difficulties for solving nonbipartite matching problems than bipartite matching problems. Flowers and blossoms are such subgraphs. A flower, defined with respect to a matching M and a root node p , is a subgraph with two components. One is a stem that is an even length alternating path that starts at the root node p and terminates at some node w . If $p = w$, the stem is empty. The other is a blossom that is an odd length alternating cycle that starts and terminates at the terminal node w of a stem and has no other node in common with the stem. w is the base of the blossom.

Figure B.2 is a simple blossom with base 5 in red and black solid lines and the stem in dotted lines. The black solid thick lines are matching edges.

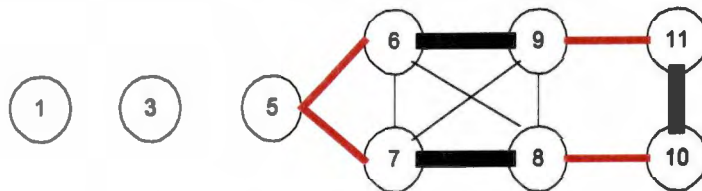


Figure B.2. A Simple Blossom

Integer Primal-Dual Programming Formulation for NMCPMP

A primal formulation for NMCPMP is considered as follows:

$$\text{Min } \sum c_{ij} x_{ij} \quad (1)$$

subject to:

$$x(i) = 1 \quad \forall i \in N' \quad (2)$$

$$Y_{\sigma}^{-}(x) \leq (|Y_{\sigma}| - 1)/2 \quad \sigma = 1 \text{ to } L \quad (3)$$

$$x_{ij} \geq 0 \text{ and integer} \quad \forall (i, j) \in E' \quad (4)$$

where:

$$x(i) = \sum x_{ij} \text{ over } j \text{ such that } (i, j) \in E'. \text{ Let } Y \subset N \text{ with } |Y| \text{ odd and } |Y| \geq 3.$$

$$Y_{\sigma}^{-}(x) = \sum x_{ij} \text{ over } i, j \text{ both } \in Y \text{ and } (i, j) \in E'.$$

Let $\{Y_1, \dots, Y_L\}$ be the set of all distinct subsets of N' of $|N'| \geq 3$.

The objective function (1) attempts to minimize the total distance of all matching pairs. Constraints (2) are perfect matching constraints. Constraints (3) are known as the matching blossom inequality constraints corresponding to $\{Y_1, \dots, Y_L\}$.

A dual formulation for NMCPMP is considered as follows:

$$\text{Max } W(\pi, \mu) = \sum_{i \in N} \pi_i - \sum_{\sigma=1}^L (|Y_\sigma| - 1)(\mu_\sigma) / 2 \quad (4)$$

subject to:

$$d_{ij}(\pi, \mu) \leq c_{ij} \quad \forall (i, j) \in E' \quad (5)$$

$$\mu \geq 0 \quad (6)$$

where:

π_i is a dual variable associated with the primal constraints (2) corresponding to node i .

μ_σ is another dual variable associated with the primal constraints (3) corresponding to the odd subset Y_σ ($\sigma = 1$ to L).

The dual variables π_i are also called original node prices. The dual variables μ_σ are known as pseudonode prices.

Let $\pi = (\pi_i)$ and $\mu = (\mu_\sigma)$, given the dual solution (π, μ) , define for each $(i, j) \in N$, there is

$$\mu^-(i, j) = \sum \mu_\sigma \text{ (over } \sigma \text{ and } Y_\sigma \text{ contains both } i \text{ and } j)$$

$$d_{ij}(\pi, \mu) = \pi_i + \pi_j - \mu^-(i, j)$$

The objective function (4) for the dual formulation is expressed as a maximum solution for the number of matching arc pairs. The complementary slackness conditions for optimality are given as:

$$x_{ij} > 0 \text{ implies } d_{ij}(\pi, \mu) = c_{ij}, \text{ for each } (i, j) \in E' \quad (6)$$

$$\mu_\sigma > 0 \text{ implies } Y_\sigma^-(x) = (|Y_\sigma| - 1)/2, \text{ for } \sigma = 1 \text{ to } L \quad (7)$$

Furthermore, $x_{ij} = 0$ implies the equality subnetwork with respect to (π, μ) and $\mu_\sigma > 0$ always implies that the associated Y_σ is the set of original nodes inside an existing pseudonode. This also guarantees that there are all but at most $(n/2)$ of the dual variable $\mu = (\mu_0)$ be 0 at every steps.

Primal-Dual Blossom Algorithm and its Solution for NMCPMP

The algorithm for NMCPMP was firstly put forward by Edmonds (1965). With his method, an algorithm can be obtained in $O(n^4)$ time. Gabow (1973) and Lawler (1976) independently discovered how to implement Edmonds's algorithm so it would run in $O(n^3)$ time. Galil, Micali, and Gabow (1982) reduced it to run in $O(n m \log n)$ time. The concept of disjoint-set operation proposed by Gabow and Tarjan (1983) further reduced the running time to be $O(\min\{n^3, n m \log n\})$, which is the best known time bound for NMCPMP.

Briefly, the primal-dual blossom algorithm discussed in the section is based on Edmonds's (1965) and Gabow's (1983) methods with disjoint-set operations. The

algorithm starts with using a search algorithm, which grows a spanning tree¹⁸ rooted at root node p . The nodes in the spanning tree are labeled nodes and unlabeled nodes. The labeled nodes are even or odd nodes¹⁹. As the search procedure proceeds, the algorithm reassigns even and odd labels to the nodes and identifies an augmentation starting at root node p . If a labeled node is reassigned and its label is different from what it already has, an even and odd alternating path with a blossom is found. The search step is suspended and replaced by a blossom shrinking and pseudonode unshrinking steps. The blossom is shrunk into a pseudonode at the base of the blossom. After the data structure is updated, the search step continues until another blossom is found out or all nodes are run out. Once we succeed in identifying an alternating path from node p to some unmatched node q , we expand the blossom represented by those pseudonodes one by one until the alternating path contains no pseudonodes. If a current equality subnetwork²⁰ contains a Hungarian forest²¹, the algorithm then goes to the dual change phase to reach another new equality subnetwork condition where at least one of the augmentation, blossom shrinking, or pseudonode unshrinking steps can be carried out.

In detail, let $\alpha_t(\pi, \mu)$ and subsets of original edges $\Gamma_t(\pi, \mu)$ and $\Delta_v(\pi, \mu)$ be defined for each current node t and current edge $(v; t)$ with respect to the present dual solution (π, μ) . More specifically,

¹⁸ A tree T is a spanning tree of G if T is a subgraph of G . Every spanning tree of a connected n -node graph G has $(n - 1)$ arcs.

¹⁹ Node i is even or odd depending on whether the number of arcs in the unique path from the root node to this node is even or odd.

²⁰ As to the Primal-dual algorithms, if the dual variable for all arcs are met with $c_{ij}^0 = c_{ij} - \mu_i - \nu_j = 0$, an equality subnetwork is obtained with respect to the dual feasible solution μ, ν .

²¹ No alternating tree can grow any further (i.e., the list of labeled nodes is empty). There are no augmenting paths in the current equality subnetwork with respect to the current matching. There are no

$$\alpha_t(\pi, \mu) = \text{Min. } \{c_{ij} - d_{ij}(\pi, \mu) : (i; j) \in E', j \text{ inside } t, i \text{ inside an EVEN node } \neq t\},$$

which is the minimum node price from adjacent edges of node t .

$\alpha_t(\pi, \mu)$ is comparable with equation (6). If $\alpha_t(\pi, \mu) = 0, x_{ij} > 0$; otherwise, also true.

$\Gamma_t(\pi, \mu) = \{(i; j) : (i; j) \in E' \text{ attains the min. in the definition of } \alpha_t(\pi, \mu)\}$, which is the edge of obtaining $\alpha_t(\pi, \mu)$.

$\Delta_v(\pi, \mu) = \{(i; j) : (i; j) \in E' \text{ attains the min. in } \min. \{c_{pg} - d_{pg}(\pi, \mu) : (p; q) \in E', p \in E', q \in E'\}\}$, which is a set of edges between v and its adjacent node t by obtaining the minimum node price.

Algorithm Statement and Implementation

The following procedures and Figure B.3 present the steps of implementing the primal-dual blossom algorithm.

Step 1: Initialize Matching

This step includes the initialization for dual variables and for an alternating forest.

First, set every node i in network G' with two alternating paths, original node prices $\pi_i (i \in N')$, and a pseudonode variable represented by pseudonode prices μ of certain pseudonodes. An initial feasible solution is set as $((\pi^0 = (\pi_i^0), \mu^0)$ where $\mu^0 = 0$ and $\pi_i^0 = (1/2)(\min. \{c_{ij} : (i, j) \in A\})$ for each $i \in N'$. Choose an initial matching in the equality subnetwork $G'(\pi^0, \mu^0)$. Initialize $d_{ij}(\pi^0, \mu^0)$ for every $(i; j) \in E'$.

blossoms which can be shrunk. There are no odd labeled pseudonodes associated with pseudonode price

Secondly, root an alternating tree at each unmatched node i to form an alternating forest. The nodes of the tree are nodes of network G' . Each node i has pointers to its father of the tree and its son, which are all initialized to point to a value of -1.

Step 2. Initialize Node Mating

There are two methods as choices for scanning and searching mates for all unmated nodes. One is called as "*greedy method*" and the other as "*heuristic method*".

The greedy method requires the ordering of all undirected edges (i, j) ($i \in N'$, $j \in N' \setminus \{i\}$) as $\{e_1, e_2, \dots, e_k, e_{k+1}, \dots, e_{n-1}, e_n\}$ where $c_1 \leq c_2, \dots, c_k \leq c_{k+1}, \dots, c_{n-1} \leq c_n$. A heap is built to store all edges. Heap operations are used to pick up one arc e_k with a smallest distance c_k among all arcs in the heap. The two end nodes i and j become mates, i.e., the mate of i is j and the mate of j is i . The arc e_k is deleted from the heap and the above steps repeat until the heap becomes empty.

Heuristic method is an improvement of the greedy method and attempts to find augmenting paths of length ≤ 3 instead of these of length ≤ 1 for the greedy method. For $\forall i \in N'$, a heap is built to store unmatched end nodes of a set of edges $E^c(i)$ which are incident in/out node i . Similarly, if the heap is not empty, the heap operations are used to find an edge $(i; j) \in E^c(i)$ and j is the end node for the edge $(i; j)$. Therefore, i and j become mates each other. If the heap is empty, it means that all edges j incident to/out i have been matched by other nodes but j . Then, the mate w for node j is found out when $w \neq i$ and an incident edge (w, v) for node w is also found out when $v \neq i$ and v has not been matched yet. Finally, an augmenting path with a length of 3 is scanned and two

pairs of matching nodes are generated, they are i and j , and w and v . The above process continues until all nodes in N' have been scanned for finding almost all such augmenting paths.

Step 3: Scan Node

A queue Q is built to store all unmatched nodes. The nodes in the queue Q are labeled as even and those not in the queue Q as unreached. If the queue Q is empty, the solution for the matching of the network G' has been obtained. If not, alternating paths and blossoms exist for the existing matching solution and the solution can be improved. Enqueue one EVEN node v' from the queue Q , iterate its incident edges from an adjacent set $E^c(v')$, and find out its mate w' .

Case 1. w' is *odd*, do nothing.

Case 2. w' is *unreached*. If $\alpha_{w'}(\pi^0, \mu^0) = 0$ and $\Gamma_{w'}(\pi^0, \mu^0)$ contains an edge incident to v' , label w' as *odd* and its mate *even*. If its mate becomes labeled as an EVEN node from an unlabeled node, for every other current node w' such that $(v'; w') \in E'$ the following needs to be done:

Find $\beta_{w'} = c_{ij} - d_{ij}(\pi^0, \mu^0)$ for some $(i, j) \in \Delta_{v', w'}(\pi^0, \mu^0)$. If

$\alpha_{w'}(\pi^0, \mu^0) < \beta_{w'}$, no change in $\alpha_{w'}(\pi, \mu)$ or $\Gamma_{w'}(\pi^0, \mu^0)$

$\alpha_{w'}(\pi^0, \mu^0) = \beta_{w'}$, replace $\Gamma_{w'}(\pi^0, \mu^0)$ by $\Gamma_{w'}(\pi^0, \mu^0) \cup \Delta_{v', w'}(\pi^0, \mu^0)$

$\alpha_{w'}(\pi^0, \mu^0) > \beta_{w'}$, change value of $\alpha_{w'}(\pi^0, \mu^0)$ to $\beta_{w'}$ and replace $\Gamma_{w'}(\pi^0, \mu^0)$

by $\Delta_{v', w'}(\pi^0, \mu^0)$.

Moreover, define v' as the father of w' . If $\alpha_w(\pi^0, \mu^0) > 0$, continue to scan next node w' .

Case 3. w' is even. If $\alpha_w(\pi^0, \mu^0) = 0$, look for current node w' containing an original v' incident to an edge in $\Gamma_w(\pi^0, \mu^0)$. Meanwhile, if one of these w' is rooted in different trees from v' . There exists an augmenting path from the root of the tree containing v' to the root of the tree of containing w' . Go to Step 4.

Case 4. w' is even. If $\alpha_w(\pi^0, \mu^0) = 0$, look for current node w' containing an original v' incident to an edge in $\Gamma_w(\pi^0, \mu^0)$. Meanwhile, v' and w' are rooted in the same tree. Edge (v', w') forms a blossom. Go to Step 5.

Step 4: Augmentation

Let $r(v')$ and $r(w')$ be two root nodes for a pair of adjacent nodes v' and w' . Accordingly, $r(v') \neq r(w')$. Trace the predecessor paths of v' and w' together with edge (v', w') and lead to the path P' , which is augmenting path between $r(v')$ and $r(w')$ by unshrinking all pseudonodes on the path P' in the current equality subnetwork. Update the matching on the base of the augmenting path P' . Repeat the procedure until the queue Q is empty. Then, go to step 7.

Step 5: Shrink Blossom

Let $r(v')$ and $r(w')$ be two root nodes for a pair of adjacent nodes v' and w' . Accordingly, $r(v') = r(w')$. Trace the predecessor paths of v' and w' and identify a blossom B . Create a new pseudonode b at root node $r(v')$ or $r(w')$, define

$E^c(b) = \cup_{k \in B} E^c(k)$. Give an even label to node b and add it to the Q . Update adjacent

links sets $E^c(j) = E^c(j) \cup \{b\}$ for each $j \in E^c(b)$. Label the nodes and adjacent links for these nodes in blossom b as unpassed blossom. The resulting network is expressed as $G^c = (N^c, A^c, C^c)$.

Let b be a new even labeled node formed as a pseudonode. Let D be the set of nodes b' on the blossom corresponding b such that $(b; b')$ is a current edge before b is formed. For each $b' \in D$ let $v_{bb'} = c_{ij} - d_{ij}(\pi^0, \mu^0)$ for any $(i; j) \in \Delta_{bb'}(\pi^0, \mu^0)$. Let $\beta_{bb'} = \min. \{v_{bb'} : t' \in D\}$. Let $X_b = \text{union of } \Delta_{bb'}(\pi^0, \mu^0) \text{ over } t' \in D \text{ satisfying } v_{bb'} = \beta_{bb'}$.

If

$$\alpha_{b'}(\pi^0, \mu^0) < \beta_{bb'}, \text{ no change in } \alpha_{b'}(\pi^0, \mu^0) \text{ or } \Gamma_b(\pi^0, \mu^0)$$

$$\alpha_{b'}(\pi^0, \mu^0) = \beta_{bb'}, \text{ replace } \Gamma_w(\pi^0, \mu^0) \text{ by } \Gamma_w(\pi^0, \mu^0) \cup X_b$$

$$\alpha_{b'}(\pi^0, \mu^0) > \beta_{bb'}, \text{ change value of } \alpha_{b'}(\pi^0, \mu^0) \text{ to } \beta_w \text{ and replace } \Gamma_b(\pi^0, \mu^0)$$

by X_b .

Also define the original base node b_0 with following values:

$$\alpha_{b_0}(\pi^0, \mu^0) = \min. \{ \beta_{bb_0} : \text{over EVEN nodes } b \text{ such that } (b; b_0) \text{ is a current edge} \}$$

$$\Gamma_{b_0}(\pi^0, \mu^0) = \text{union of } X_b \text{ over } b \text{ attaining the minimum in the definition of}$$

$$\alpha_{b_0}(\pi^0, \mu^0)$$

$$\Delta_{bb_0}(\pi^0, \mu^0) = X_b$$

Repeat the procedure until the queue Q is empty. Then, go to step 7.

Step 6: Unshrunk Pseudonode

Unshrink a pseudonode b' that is odd labeled with the associated pseudonode price $\mu_\sigma = 0$ in the present dual solution. Revise the set of current matching edges and the set of current nodes accordingly. Also, revise adjacent link set $E^c(j') = E^c(j) \setminus \{b'\}$ for each $j' \in E^c(b')$. Add new even labeled nodes in the blossoms corresponding to the unshrunk pseudonodes to the queue Q .

Let b' be a pseudonode which has been unshrunk into v' . Let node w' on the blossom corresponding to v' and edges $(v'; w')$ incident at it after the unshrinking. Compute $\alpha_{w'}(\pi^0, \mu^0)$, $\Gamma_{w'}(\pi^0, \mu^0)$, and $\Delta_{w'}(\pi^0, \mu^0)$ using their definitions. Also, define $v_{v', w'} = c_{ij} - d_{ij}(\pi^0, \mu^0)$ for any $(i; j) \in \Delta_{v', w'}(\pi^0, \mu^0)$, $\beta_{v'} = \min. \{v_{v', w'} : w' \text{ is a EVEN labeled node on the blossom corresponding to } b'\}$, $X_{v'} = \text{union of } v_{v', w'}$ over all w' attaining the minimum in the definition w' . If

$$\alpha_{v'}(\pi^0, \mu^0) < \beta_{v'}, \text{ no change in } \alpha_{v'}(\pi^0, \mu^0) \text{ or } \Gamma_{v'}(\pi^0, \mu^0)$$

$$\alpha_{v'}(\pi^0, \mu^0) = \beta_{v'}, \text{ replace } \Gamma_{v'}(\pi^0, \mu^0) \text{ by } \Gamma_{v'}(\pi^0, \mu^0) \cup X_{v'}$$

$$\alpha_{v'}(\pi^0, \mu^0) > \beta_{v'}, \text{ change value of } \alpha_{v'}(\pi^0, \mu^0) \text{ to } \beta_{v'} \text{ and replace } \Gamma_{v'}(\pi^0, \mu^0)$$

by $X_{v'}$.

Repeat this procedure until there are no pseudonodes that are ODD labeled associated with $\mu_\sigma = 0$. Go back to step 3.

Step 7: Change Dual Solutions

If there are no alternating paths that can grow any further, no blossoms which can be shrunk, and no odd labeled pseudonodes associated with pseudonode price $\mu_\sigma = 0$, it

implies that the present matching is a maximum number matching in the current equality subnetwork but may not be a minimum cost matching.

$$\delta_1 = \min.\{\alpha_t(\pi^0, \mu^0) : t \text{ is an unlabeled current node}\}$$

$$\delta_2 = \min.\{\frac{1}{2}\alpha_t(\pi^0, \mu^0) : t \text{ an even labeled current node}\}$$

$$\delta_3 = \min.\{\frac{1}{2}\mu_\sigma : \sigma \text{ s.t. } Y_\sigma \text{ is the set of original nodes inside a current odd}$$

labeled pseudonodes\}

$$\delta = \min.\{\delta_1, \delta_2, \delta_3\}$$

If $\delta = +\infty$, go to step 8. If δ is finite, define the new dual variable to be $\hat{\pi} =$

$(\hat{\pi}_i), \hat{\mu} = (\hat{\mu}_\sigma)$. Then

$$\alpha_t(\hat{\pi}, \hat{\mu}) = \begin{cases} \alpha_t(\pi^0, \mu^0) - \delta & \text{if } t \text{ is an unlabeled node} \\ \alpha_t(\pi^0, \mu^0) - 2\delta & \text{if } t \text{ is an even unlabeled node} \\ \alpha_t(\pi^0, \mu^0) & \text{if } t \text{ is an odd unlabeled node} \end{cases}$$

$$\Gamma_i(\hat{\pi}, \hat{\mu}) = \Gamma_i(\pi^0, \mu^0), \Delta_n(\hat{\pi}, \hat{\mu}) = \Delta_n(\pi^0, \mu^0)$$

$$\hat{\mu}_\sigma = \begin{cases} \mu_\sigma + 2\delta & \text{if } Y_\sigma \text{ is the set of original nodes in an even labeled pseudonode} \\ \mu_\sigma + \delta & \text{if } Y_\sigma \text{ is the set of original nodes in an odd labeled pseudonode} \\ \mu_\sigma & \text{otherwise} \end{cases}$$

Rebuild queue Q and add all even labeled nodes to it. If $\delta = \delta_3$, go to step 6. If

$\delta < \delta_3$, go to step 3. If queue Q is empty and finite minimum cost for matching nodes is achieved, go to step 8 or step 9.

Step 8: Unfeasibility

There exists no minimum cost perfect matching for G' .

Step 9: Optimality

There exists a minimum cost perfect matching for G' , stop.

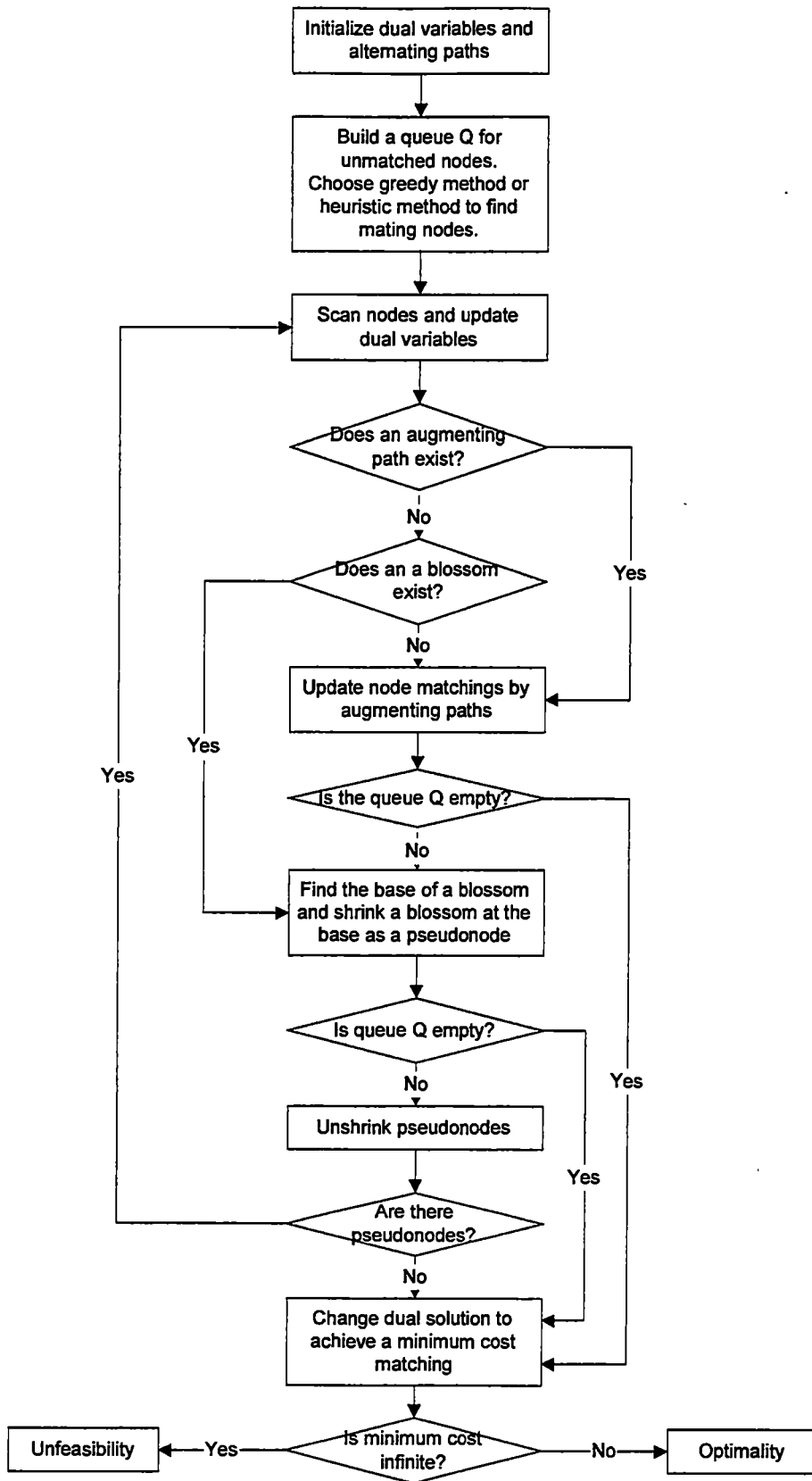


Figure B.3. A Flow Diagram of Primal-Dual Blossom Algorithm

Two Examples of Primal-Dual Blossom Algorithms

Case 1: A Simple Derived Matching Network without Blossoms

The CARP network in Figure B.4 includes four odd-degree nodes. Let $G'(N', E', C')$ be a derived matching network. Its node set N' is $\{n_2, n_3, n_4, n_5\}$, which is a set of the odd-degree nodes. Its edge node set E' is $\{e_{23}, e_{24}, e_{25}, e_{34}, e_{35}, e_{45}\}$, which is a new set of edges whose cost set C' is defined a set of shortest paths as $\{c_{23}, c_{24}, c_{25}, c_{34}, c_{35}, c_{45}\}$. Figure B.4 shows the derived matching network $G'(N', E', C')$ from Figure 3.2.

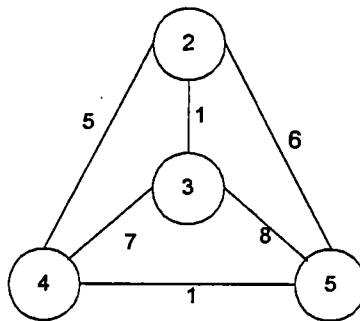


Figure B.4. A Derived Matching Network²² without Blossoms

Step 1: Initialize values associated with the edges of $G'(N', E', C')$. $\pi_i^0 = (1/2)(\min. \{c_{ij}: (i, j) \in A\})$. $\mu^-(i, j)$ and D_{ij} are all initialized as 0 and 1 respectively.

²² The number on the edge indicates the length.

Edge (i, j)	π_i	π_j	$\mu^-(i, j)$	D_{ij}
2, 3	0.5	0.5	0	1
2, 4	0.5	0.5	0	1
2, 5	0.5	0.5	0	1
3, 4	0.5	0.5	0	1
3, 5	0.5	0.5	0	1
4, 5	0.5	0.5	0	1

Then, values associated with the nodes of $G'(N', E', C')$ are initialized. Since no labeled nodes and pseudonodes exist in this step, initialize $\alpha_i(\pi, \mu)$ as $+\infty$. $\Gamma_i(\pi, \mu)$ for all $i \in N'$ and $\Delta_{vi}(\pi, \mu)$ for all $((v; i) \in E')$ are ignored. An alternating forest composed of alternating trees is formed as below.

Node i	$\alpha_i(\pi, \mu)$	Alternating Trees		
		Father Node	Son Node	Tree Size
		ID	ID	
2	$+\infty$	-1	-1	1
3	$+\infty$	-1	-1	1
4	$+\infty$	-1	-1	1
5	$+\infty$	-1	-1	1

Step 2: When applying “greedy method” and “heuristic method”, edges and nodes have the following attributes.

Edge (i, j)	Length $c(i, j)$
2, 3	1
4, 5	1
2, 4	5
2, 5	6
3, 4	7
3, 5	8

Node i	Mate(i)
2	INVALID
3	INVALID
4	INVALID
5	INVALID

For the “*greedy method*”, the heap operation firstly picks up edge(2, 3) with the shortest distance between node 2 and 3 become mates, i.e, $\text{mate}(2) = 3$, $\text{mate}(3) = 2$. Similarly, edge (4, 5) is selected and $\text{mate}(4) = 5$, $\text{mate}(5) = 4$.

Different from the “*greedy method*”, the “*heuristic method*” chooses a node 2 at first and uses heap operation to pick an edge starting from 2 with smallest length and

ending to a node without a mate, which is edge (2, 3). Therefore, $\text{mate}(2) = 3$ and $\text{mate}(3) = 2$. Next node is 4 and edge(4, 5) is found with $\text{mate}(4) = 5$, $\text{mate}(5) = 4$.

Once all nodes have found mates, the two methods stop and show the following result.

Node i	Mate(i)
2	3
3	2
4	5
5	4
Total Distance	2

Since all nodes have mates, no further steps are needed and the matching solution is found. The computation of the resulting lower bounds are presented in Section 3.3. A second example is given to clarify the procedures of dealing with flowers in a matching network.

Case 2: A Complicated Matching Network with Blossoms

Figure B.5 shows another $G'(N', E', C')$ as a matching network with blossoms.

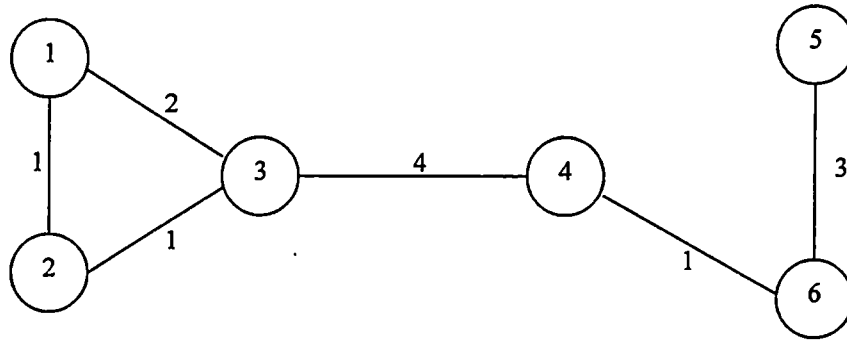


Figure B.5 A Matching Network with Blossoms

Step 1: First, initialize values associated with the edges of $G'(N', E', C')$.

Edge (i, j)	π_i	π_j	$\mu^-(i, j)$	d_{ij}
1, 2	0.5	0.5	0	1
1, 3	0.5	0.5	0	1
2, 3	0.5	0.5	0	1
3, 4	0.5	0.5	0	1
4, 6	0.5	0.5	0	1
5, 6	1.5	0.5	0	2

Then, values associated with the nodes in $G'(N', E', C')$ are initialized.

Node i	$\alpha_i(\pi, \mu)$	pre(i)	Alternating Trees		
			Father Node ID	Son Node ID	Tree Size
1	$+\infty$	INVALID	-1	-1	1
2	$+\infty$	INVALID	-1	-1	1
3	$+\infty$	INVALID	-1	-1	1
4	$+\infty$	INVALID	-1	-1	1
5	$+\infty$	INVALID	-1	-1	1
6	$+\infty$	INVALID	-1	-1	1

Step 2: When applying the “*greedy method*”, edges and nodes have following attributes.

Edge (i, j)	Length $c(i, j)$
1, 2	1
1, 3	2
2, 3	1
3, 4	4
4, 6	1
5, 6	3

Node i	Mate(i)	Pre(i)
1	INVALID	INVALID
2	INVALID	INVALID
3	INVALID	INVALID
4	INVALID	INVALID
5	INVALID	INVALID
6	INVALID	INVALID

The result shows that $\text{mate}(1) = 2$, $\text{mate}(2) = 1$ and $\text{mate}(4) = 6$, $\text{mate}(6) = 4$, which is not the maximum number of matching. Continue Step 3.

Node i	Mate(i)	Pre(i)	Label(i)	α_i	Γ_i	Δ_{ij}
1	2	INVALID	UNREACHED	0	Edge(1, 2)	Edge(1, 2)
2	1	INVALID	UNREACHED	0	Edge(1, 2)	Edge(1, 2)
3	INVALID	INVALID	INVALID	$+\infty$	No	No
4	6	INVALID	UNREACHED	0	Edge(4, 6)	Edge(4, 6)
5	INVALID	INVALID	INVALID	$+\infty$	No	No
6	4	INVALID	UNREACHED	0	Edge(4, 6)	Edge(4, 6)

Step 3. A new queue $Q = \{\text{node 3, node 5}\}$

Dequeue(Q) = node 3.

Case 2. For edge (3, 4), $\beta_4 = c_{34} - d_{34} = 4 - 1 = 3$. Set $\text{Pre}(4) = 3$ and $Q = \{\text{node 5, node 6}\}$. No change is for node 4.

Node i	Mate(i)	Pre(i)	Label(i)	α_i	Γ_i	Δ_{ii}
1	2	INVALID	UNREACHED	0	Edge(1, 2)	Edge(1, 2)
2	1	INVALID	UNREACHED	0	Edge(1, 2)	Edge(1, 2)
3	INVALID	INVALID	EVEN	$+\infty$	No	No
4	6	3	ODD	0	Edge(4, 6)	Edge(4, 6)
5	INVALID	INVALID	INVALID	$+\infty$	No	No
6	4	INVALID	EVEN	0	Edge(4, 6)	Edge(4, 6)

Case 2. For edge (2, 3), $\beta_2 = c_{23} - d_{23} = 1 - 1 = 0$. Set Pre(2) = 3 and $Q = \{\text{node 5, node 4, node 1}\}$. There are changes for node 2.

Node i	Mate(i)	Pre(i)	Label(i)	α_i	Γ_i	Δ_{ii}
1	2	INVALID	EVEN	0	Edge(1, 2)	Edge(1, 2)
2	1	3	ODD	0	Edge(1, 2) \cup Edge(2, 3)	Edge(1, 2)
3	INVALID	INVALID	EVEN	$+\infty$	No	No
4	6	3	ODD	0	Edge(4, 6)	Edge(4, 6)
5	INVALID	INVALID	INVALID	$+\infty$	No	No
6	4	INVALID	EVEN	0	Edge(4, 6)	Edge(4, 6)

Case 3. for edge (1, 3), a blossom composed of node 1, node 2 and node 3 is found since two paths (node 3 – node 2 – node 1, node 3 – node 2) share with the same base node 3. Go to Step 5 for blossom shrinking with node 7 as a pseudonode. The

network is changed as follows:

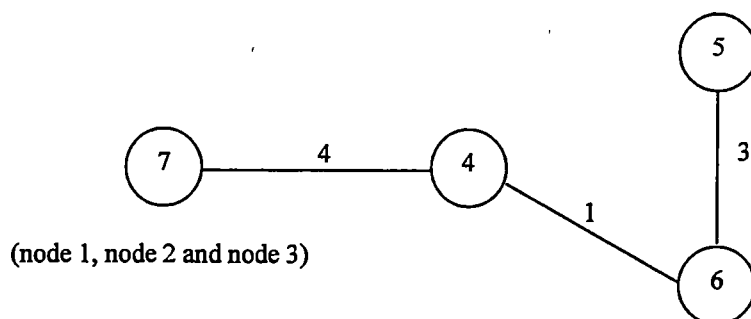


Figure B.6. A Matching Network with a Shrunk Blossom and a Pseudonode

$$\beta_7 = \text{Min. } \{ \beta_1, \beta_2 \} = \text{Min. } \{ c_{13} - d_{13}, c_{23} - d_{23} \} = \text{Min. } \{ 2 - 1, 1 - 1 \} = 0$$

$$X_7 = \{ \text{Edge } (2, 3) \}$$

$$Q = \{ \text{node 5, node 4, node 1, node 2} \}$$

$$Y_1^-(7) = x_{12} = 1$$

$$(|Y_1^-| - 1) / 2 = (3 - 1) / 2 = 1, \text{ for } \sigma = 1.$$

Because $Y_1^-(7) = (|Y_1^-| - 1) / 2$, $\mu_1 > 0$ for pseudonode 7.

Values of node 3 are changed and a new pseudonode 7 is added with even label.

Node i	Mate(i)	Pre(i)	Label(i)	α_i	Γ_i	Δ_i
1	2	INVALID	EVEN	0	Edge(1, 2)	Edge(1, 2)
2	1	3	ODD	0	Edge(1, 2) \cup Edge(2, 3)	Edge(1, 2)
3	INVALID	INVALID	EVEN	0	Edge (2, 3)	Edge (2, 3)
4	6	3	ODD	0	Edge(4, 6)	Edge(4, 6)
5	INVALID	INVALID	INVALID	$+\infty$	No	No
6	4	INVALID	EVEN	0	Edge(4, 6)	Edge(4, 6)
7	INVALID	INVALID	EVEN	$+\infty$	No	No

Again, return to Step 3. Dequeue Q {node 5, node 4, node 1, node 2} = node 5.

Case 3. For Edge (5, 6), node 6 is labeled as EVEN and node 5 and node 6 are on the different trees. An augmentation path is found, which is node 5 – node 6 – node 4 – node 3. The labels are changed for these nodes. Readjust the mates for nodes 5, 6, 4, 3 according to the augmentation path.

Node i	Mate(i)	Pre(i)	Label(i)	α_i	Γ_i	Δ_{ij}
1	2	INVALID	EVEN	0	Edge(1, 2)	Edge(1, 2)
2	1	3	ODD	0	Edge(1, 2) \cup Edge(2, 3)	Edge(1, 2)
3	4	5	ODD	0	Edge (2, 3)	Edge (2, 3)
4	3	5	EVEN	0	Edge(4, 6)	Edge(4, 6)
5	6	5	EVEN	$+\infty$	No	No
6	5	5	ODD	0	Edge(4, 6)	Edge(4, 6)
7	INVALID	5	ODD	$+\infty$	No	No

Step 6. Since pseudonode 7 is ODD labeled but with $\mu_1 > 0$, there is no need of pseudonode unshrinking.

Dequeue Q {node 4, node 1, node 2} = node 4.

Case 1. Node 3 on Edge (3, 4) and node 6 on Edge (4, 6) are ODD labeled. The network does not change.

Dequeue Q {node 1, node 2} = node 1.

Case 1. Node 3 on Edge (1, 3) and node 2 on Edge (1, 2) are ODD labeled. The network does not change.

Dequeue Q {node 2} = node 2.

Case 1. Node 3 on Edge (2, 3) is ODD labeled. The network does not change.

Case 4. Node 1 on Edge (1, 2) is EVEN labeled and node 1 and node 2 are the same tree. An augmentation path node 3 – node 2 – node 1 is found. Readjust mates for nodes 1, 2, 3 according to the augmentation path.

Node i	Mate(i)	Pre(i)	Label(i)	α_i	Γ_i	Δ_{ii}
1	2	3	ODD	0	Edge(1, 2)	Edge(1, 2)
2	1	3	EVEN	0	Edge(1, 2) \cup Edge(2, 3)	Edge(1, 2)
3	4	5	ODD	0	Edge (2, 3)	Edge (2, 3)
4	3	5	EVEN	0	Edge(4, 6)	Edge(4, 6)
5	6	5	EVEN	$+\infty$	No	No
6	5	5	ODD	0	Edge(4, 6)	Edge(4, 6)
7	INVALID	INVALID	ODD	$+\infty$	No	No

Queue Q becomes empty, go to Step 7 for dual solution check and change.

Step 7. Without consideration of pseudonode 7, $\alpha_i = 0$ and $\mu_1 > 0$, which meet the dual condition. Also, all $x(i) = 1$ ($i = 1, 2, 3, 4, 5, 6$) and $Y_1^-(7) = (|Y_1| - 1)/2$ for $\sigma = 1$, which meet the primal condition. Moreover, the complementary condition is met. Go to Step 9 and an optimum solution is found.

APPENDIX C – AVENUE Pseudocodes for SWRoute Interface Design

Interface Functions for Generating Base road Network

Select Base Network

Event SelectBaseNet.Click()

Begin

If name of a node theme $nTheme$ = name of a line theme $lTheme$ **then**

 Add $nTheme$ to nLIST;

 Add $lTheme$ to lLIST;

End

Set $_BASE\ NETWORK^l = a \in nTheme$;

Set $_BASENODE = b \in lThem$;

Set a symbol type for $_BASENETWORK$ and $_BASENODE$ shapes;

End;

Build Base Network

Event BuildBaseNetwork.Click()

Begin

Set *fnLIST* = {"*Vehicle*", "*Date*", "*Two_way*", "*roadname*", "*weight*"};

Compare each field *f* in _BASENETWORK attribute table with *fnList*

Set the name of *f* "*Vehicle*" = "*Driver0*";

Set the name of *f* "*Date*" = "*Day0*";

Set the name of *f* "*Two_way*" = "*Two-way*";

Add field "*roadname*" and Set field "*roadname*" =

"*Fdpre*" + "*Fname*" + "*Ftype*";

Add field "*weight*" and Set field "*weight*" = field "*Length*" divided
by the total length of the same road;

End;

Interface Functions for Generating Subnetworks

Create A Partition Network Workspace

Event CreateTargetNetwork.Click()

Begin

Set *tn* = a Target Network and its directory;

If (*tn*'s directory Is Existing) = True **then**

 Delete every folder and file in that directory;

Else

 Create a new directory;

End;

Copy _BASENETWORK to _TARGETLN and add empty fields

{*“driver”*, *“day”*};

Copy _BASENODE to _TARGETPT and add empty fields {*“driver”*,
“day”};

Add _TARGETLN and _TARGETPT to the project interface;

Set a symbol type for _TARGETLN and _TARGETPT shapes;

End;

Open A Partition Network Workspace

Event OpenTargetNetwork.Click()

Begin

Remove _TARGETLN and _TARGETPT from the project interface;

Go to another partition network workspace and set it as _TARGETLN and
_TARGETPT workspace;

Add _TARGETLN and _TARGETPT to the project interface;

Set a symbol type for _TARGETLN and _TARGETPT shapes;

End;

Delete A Partition Network

Event spacedelete.Click()

Begin

Remove _TARGETLN and _TARGETPT from the project interface;

Delete everything in their workspace;

End;

Set Driver Partition Interface

Event PartitionDriverNetwork()

Begin

Display a summary report about the information "Dirver0" in the
_TARGETNET attribute table;

If the toolbar for weekday partition existing then

Set invisible = true;

If the toolbar for driver partition existing then

Remove the toolbar;

If the number of drivers $\geq 10^1$ then

Return;

Set *driverIndex* = 1;

Add a driverpartition toolgroup at the right end of the toolbar;

For each driver *i* do

Begin

Add a tool for driver *i* to the driver partiton toolgroup;

Set Tool.Icon = *driverIndex* number;

Set Tool.Help = driver code number;

Set Tool.Apply.Script = "driverAdd.Apply";

Set Tool.Tag = "driver";

Set Tool.ObjectTag¹ = *driverIndex* + driver code;

End;

(continued)

driverIndex = *driverIndex* + 1;

End;

Add a driver partition deletion tool to the toolgroup;

With the deletion tool

Begin

Set Tool.Click.Script = "View.SelectTool";

Set Tool.Apply.Script = "driverDelete.Apply";

Set Tool.Update.Script = "View.HasDataUpdate";

Set Tool.Tag = "driver";

End;

End;

Add buttons "Driver Reference On/Off", "Legend On/Off", and "Report"

to the view;

End;

Show(Hide) Last/Base Driver Partition

Event DriverBasePartition.Click()

Begin

If Menu Label = "Hide Last/Base Driver Partition" then

 Delete all graphics within the project view area;

End;

If select last driver partition then

 Set driver field = "Driver";

Else If select base driver partition then

 Set driver field = "Driver0";

End;

For each link with non-zero demand do

Begin

 Make a graphic shape G for the link;

 Set G.ObjectTag = "driver,"+tpID+featureID+demand;

 Set G.Symbol;

 Add G to _TARGETLN;

End;

End;

(continued)

Event DriverBasePartition.Update()

Begin

If _TARGETLN \neq NULL **then**

 Set _TARGETLN.Enabled = True;

End;

End;

Write Drivers To Partition Network

Event WriteDriver.Click()

Begin

For each graphic shape do

Begin

Get its featureID in _TARGETLN attribute table;

Get driver code number from its object tag;

Set field "Driver" with the driver code number;

End;

End;

End;

Driver Reference On(Off)

Event DriverReference.Click()

Begin

Set legends for _BASENETWORK by field "Driver0";

End;

End;

Set Driver-Weekday Partition Interface

Event PartitionDayNetwork()

Begin

Display a summary report about the information from “Dirver0” and
“Day0” in the _TARGETNET attribute table;

If the toolbar for driver partition existing then

Set invisible = true;

If the toolbar for day partition existing then

Remove the toolbar;

Set *dayIndex* = 1;

Add a daypartition toolgroup at the right end of the toolbar;

For each weekday *i* do

Begin

Add a tool for *i* to the day partiton toolgroup;

Set Tool.Icon = Weekday;

Set Tool.Apply.Script = “dayAdd.Apply”;

Set Tool.Tag = “day”;

Set Tool.ObjectTag = *dayIndex* + weekday;

End;

(continued)

DayIndex = *dayIndex* + 1;

End;

Add a day partition deletion tool to the toolgroup;

With the deletion tool

Begin

Set Tool.Click.Script = "View.SelectTool";

Set Tool.Apply.Script = "dayDelete.Apply";

Set Tool.Update.Script = "View.HasDataUpdate";

Set Tool.Tag = "day";

End;

End;

Add buttons "Day Reference On/Off", "Legend On/Off", "Report", and a dropdown box to the view;

End;

Show(Hide) Last/Base Day Partition

Event DayBasePartition.Click()

Begin

If Menu Label = "Hide Last/Base Day Partition" **then**

 Delete all graphics within the project view area;

End;

If select last day partition **then**

 Set driver field = "Driver";

 Set day field = "Day";

Else If select base driver partition **then**

 Set driver field = "Driver0";

 Set day field = "Day0";

End;

For each link with non-zero demand **do**

Begin

 Make a graphic shape G for the link;

 Set G.ObjectTag = "day,"+tpID+featureID+demand;

 Set G.Symbol;

 Add G to _TARGETLN;

End;

End;

(continued)

Event DayBasePartition.Update()

Begin

If _TARGETLN \diamond NULL **then**

 Set TARGET.Enabled = True;

End;

End;

Write Driver-Weekdays To Partition Network

Event WriteDay.Click()

Begin

For each graphic shape **do**

Begin

 Get its featureID in _TARGETLN attribute table;

 Get weekday from its object tag;

 Set field "Day" with weekday;

End;

End;

End;

Driver-Weekday Reference On(Off)

Event DayReference.Click()

Begin

Set legends for _BASENETWORK by field "Day0";

End;

End;

Build Subnetworks

Event BuildSubnetworks()

Begin

If _TARGETLN attribute field "Driver" or "Day" is empty **then**

 Can not make subnetworks and return;

End;

 Set all _TARGETLN attribute fields invisible except "Fnode#", "Tnode#", "Length", "tpID", "Fromlong", "Fromlat", "Tolong", "Tolat", "Two-way", "Demand", "Driver", and "Day";

 Adjust field order as above;

 Set all _TARGETPT attribute fields invisible except "nodeID"

For each driver do

For each weekday do

 Export _TARGETLN attribute table to an ASCII comma delimited file for the driver on a weekday;

End;

End;

 Generate an ASCII file with sorted unique node ID from _TARGETPT;

 Generate forward star network data structure for 20 subnetworks;

 Set all fields in _TARGETLN and _TARGETPT visible;

End;

Interface Functions for Generating a Seed Node Set

Add Garaga/Landfill/Depot

Event NodeAdd.Click()

Begin

Select a tool with a tag "ADDNODE";

End;

Event NodeAdd.Apply()

Begin

Get a user point p by clicking;

If p is not at a node location of `_TARGETPT` **then**

Return;

End;

If p is at a node location with more than one `_TARGETPT` node **then**

Display a tpID list to select one node to be used;

End;

If p already exists in the seed node set **then**

Display a message "add another seed node at this location?";

If answer is no **then**

Return;

End;

End;

Display "Adding Garage/Lanfill/Depot" dialog box with a tpID which has already shown;

(continued)

Click "Ok" of the dialog box, usedID increment 1, a stop node appearing on the interface and a new node record added to the attribute table;

End;

Edit Stop Attributes

Event NodeEdit.Click()

Begin

Select a tool with tag "IDNODE";

End;

Event NodeEdit.Apply()

Begin

Get a user point p ;

If p does not exists as a seed node then

Return;

End;

If p is at a location with more than one seed node then

Display a tpID list to select one seed node for editing;

End;

Display "Edit Depot Attributes" Dislog Box and modified attributes are written to the seed node attribute table;

End;

Delete Stops

Event NodeDelete.Click()

Begin

Select a tool with tag "DELNODE"

End;

Event NodeDelete.Apply()

Begin

Get a user point p ;

If p is not a seed node then

Return;

End;

If p exists at a location with more than one seed node then

Display a tpID list to select one for deletion;

End;

Delete that node from the interface and its attribute record from the
attribute table;

End;

Move Depots

Event NodeMove()

Begin

Select a tool with tag "MOVENODE";

End;

Event NodeMove.Apply()

Begin

Get a user point m as a from node;

Check its seed node existence and multiple selection;

Get another user point n as a to node;

Check its seed node existence and multiple selection;

Swap attribute records for the two nodes;

End;

Interface Functions for Route Generating and Reporting

Build New Routes

Event CreateGraphicTour.Click()

Begin

Display a dialog box to collect information for route configuration;

If no seed nodes existing for a selected driver then

Return;

End;

If no seed nodes existing for all selected driver and his weekdays then

Return;

End;

For each existing driver-weekday do

Generating routes;

Generating shortest paths;

End;

End;

Show/Hide New Routes

Event ShowGraphicTour.Click()

Begin

Go to route directory and make a list of all route files and its associated shortest path files;

Display a multiple selection list box with the list;

Delete all route themes in the interface;

For each route selection do

Open the shortest path file and add a new record from a link on a shortest path to a new shape file. The shape file keeps all the information used for a route object;

Open the route file and append new records from each link in a route to the same shape file;

For each newly generated shape file do

Add route shape files to the interface;

Display the tour and shortest paths with specific graphic shapes and attach these graphic shape with route object tags;

End;

End;

Report Current Routes

Event RouteInfo.Click()

Begin

Select a tool with tag "ROUTEINFO";

End;

Event RouteInfo.Apply()

Begin

Select a graphic shape *g*;

If *g* is on the shortest path or on a link with no trash **then**

Remove all graphic shapes from the interface;

Return;

End;

Get the object tag for *g*;

Display a notepad report and a graphic report only for one route and its related shortest path where the trash linked to *g* is collected;

End;

APPENDIX D – C Source Codes for Solving Capacitated Arc Routing Problems

```
/*-----*/
/*
/* TITLE:
/*   netlib.h
/*
/* DESCRIPTION:
/*   A network utility header file
/*
/* ORIGIN:
/*   Programmer: Xiaohong Xin
/*   Date:      20 February 1998
/*   Platform:  Windows 95
/*   Compiler:  Borland C++ 5.0
/*
/*-----*/
#ifndef _NETLIB_H
#define _NETLIB_H

typedef struct header_t
{
    int      max;
    int      size;
    int      num;
    void     **v;
} header_t;

extern FILE* Open_File (char*, char*);
extern header_t* Init_Header (int);
extern void* New_Element (int);
extern void Append_Element (header_t*, void*);
extern void Free_Data(header_t *);

#endif
```



```

/*=====*/
/*
/* TITLE:
/*     network.h
/*
/*
/* DESCRIPTION:
/*     A network header file
/*
/*
/* ORIGIN:
/*     Programmer: Xiaohong Xin
/*     Date:      20 February 1998
/*     Platform:  Windows 95
/*     Compiler:  Borland C++ 5.0
/*
/*=====*/

#ifndef _NETWORK_H
#define _NETWORK_H

#define SIZEOFBUFFER 1024
enum passFlag {PASS=0, NOPASS, PSEUDO, BLOSSOM};

typedef struct location
{
    long int    lon;
    long int    lat;
} location_t, *ptr_location_t;

typedef struct fstar
{
    int*        anode;
    int*        bnode;
    int*        cnode;
} fstar_t, *ptr_fstar_t;

typedef struct NODE
{
    location_t  aLocation;
    int         outLinks;
    int         inLinks;
    int         undirectedLinks;
    int         nodeID;
} node_t, *ptr_node_t;

typedef struct NODES
{
    int         max;
    int         size;
    int         num;
    node_t**   v;
} nodes_t, *ptr_nodes_t;

typedef struct LINK
{
    int         anode;
    int         bnode;
    int         arc_id;
    int         indexID;
    int         legFlag;
    int         uniqueFlag;
    float       trashamount;
    int         startTime;
    int         endTime;
    float       distance;
    float       time;
    short int   two_way; /*two_way=2, one_way=1 */
    int         passFlag;
    location_t  org_node;
    location_t  dst_node;
} link_t, *ptr_link_t;

typedef struct LINKS
{
    int         max;
    int         size;
    int         num;
    link_t**   v;
}

```

```

} links_t, *ptr_links_t;

typedef struct NET
{
    nodes_t*          nodes;
    links_t*          links;
    fstar_t*          fstar;
} net_t, *ptr_net_t;

typedef struct load
{
    int      fnode;
    int      tnode;
    int      arc_id;
    double   distance;
    float    time;
    location_t org; /* origin location */
    location_t dst; /* destination location */
    short int two_way; /*two_way=2, one_way=1 */
    int      trashamount;
} load_t, *ptr_load_t;

typedef struct loads
{
    int      max;
    int      size;
    int      num;
    load_t   **v;
} loads_t, *ptr_loads_t;

extern void      NetBuild(char *, char *, char *);
extern load_t*  New_Load (int, int, float, int, long int, long int, long int, long int, short int,
float);
extern node_t*  New_Node (int);
extern ptr_loads_t  Get_Loads (FILE *);
extern void     Build_Nodes (ptr_net_t, FILE*);
extern ptr_link_t  CopyLink(ptr_link_t);
extern ptr_link_t  ReverseLink(ptr_net_t, ptr_link_t);
extern int        Compare_Links (const void *, const void *);
extern int        Compare_Nodes (const void *, const void *);
extern void       Build_Forward_Star (ptr_net_t);
extern void       Add_To_Net (ptr_loads_t, ptr_net_t);
extern void       Write_To_Net (ptr_net_t, FILE*);
extern ptr_net_t  Init_Network (void);
extern int        Even_Odd(int);
#endif

```

```

/*=====*/
/*
/* TITLE:
/*     path.h
/*
/*
/* DESCRIPTION:
/*     A path header file
/*
/*
/* ORIGIN:
/*     Programmer: Xiaohong Xin
/*     Date:      20 February 1998
/*     Platform:  Windows 95
/*     Compiler:  Borland C++ 5.0
/*
/*=====*/
#ifndef _PATH_H
#define _PATH_H

#define TRUE 1
#define FALSE 0
#define INVALID -10
#define FIXED -20
#define TRAVERSED -30
#define UNIQUE 100
#define END -40
#define INFINITY 9.99E+8

typedef struct PATHNODE
{
    int fornode;
    int label;
    int parent;
    int arc_id;
    int indexID;
    float distance;
    int dir;
}pathnode_t, *ptr_pathnode_t;

typedef struct PATHNODES
{
    int max;
    int size;
    int num;
    pathnode_t **v;
}pathnodes_t, *ptr_pathnodes_t;

typedef struct ODNODE
{
    int * orig;
    int * dest;
}odnodes_t, *ptr_odnodes_t;

extern ptr_odnodes_t Init_OD(int);
extern void ReadNet(FILE *, ptr_net_t);
extern ptr_odnodes_t ReadOD(FILE *, int);
extern int FindPath(int, int, ptr_net_t, ptr_pathnodes_t);
extern void UpdateTree(int, int, ptr_net_t, ptr_pathnodes_t);
extern ptr_pathnodes_t ClearOutTree(int, ptr_net_t);
extern int WritePath(int, int, FILE *, ptr_net_t, ptr_pathnodes_t);
extern int WriteOutPath(int, int, ptr_net_t, ptr_pathnodes_t);
extern void Free_Shtpath(ptr_net_t, ptr_pathnodes_t, ptr_odnodes_t);

#endif

```

```

/*=====*/
/*
/* TITLE:
/*   queue.h
/*
/* DESCRIPTION:
/*   A queue header file
/*
/* ORIGIN:
/*   Programmer: Xiaohong Xin
/*   Date:      20 February 1998
/*   Platform:  Windows 95
/*   Compiler:  Borland C++ 5.0
/*
/*=====*/
#ifndef _QUEUE_H
#define _QUEUE_H

#ifndef _BOOL_
#define _BOOL_
typedef enum {false=0, true} BOOLEANs;
#endif

typedef struct
{
    int head, tail, size;
    int *data;
} Queue;

extern void Barf(char *);
extern Queue *MakeQueue(int);
extern void InitQueue(Queue *);
extern int Dequeue(Queue *);
extern void Enqueue(Queue *, int);
extern int QSize(Queue *);
BOOLEANs QueueEmpty(Queue *);
extern void DestroyQueue(Queue *);

#endif

```

```

/*=====*/
/*
/* TITLE:
/*     lld.h
/*
/* DESCRIPTION:
/*     A doubly linked list header file
/*
/*
/* ORIGIN:
/*     Programmer: Xiaohong Xin
/*     Date:      20 February 1998
/*     Platform:  Windows 95
/*     Compiler:  Borland C++ 5.0
/*
/*=====*/
#ifndef __LLD_H
#define __LLD_H

#ifndef _BOOL_
#define _BOOL_
typedef enum {false=0, true} BOOLEANS;
#endif

typedef struct HEAD
{
    ptr_link_t    headlink;
    int           headnode;
    float         saving_Head_distance;
} head_t, *ptr_head_t;

typedef struct TAIL
{
    ptr_link_t    taillink;
    int           tailnode;
    float         saving_Tail_distance;
} tail_t, *ptr_tail_t;

typedef struct HEAD_TAIL
{
    int           NumHeads;
    int           NumTails;
    head_t **     mergeHeadNodes;
    tail_t **     mergeTailNodes;
} head_tail_t, *ptr_head_tail_t;

typedef struct _lnode
{
    void * data; /*a current point */
    struct _lnode *next;
    struct _lnode *prev;
} *lnode;

typedef struct _list
{
    lnode tail, current, head;
    float time;
    float distance;
    float demand;
    int numTraversed;
    head_tail_t * mergeNodes;
    int length;
} *LLD;

extern LLD createLLD(void);
extern void destroyLLD(LLD *);
extern void destroyLegs(LLD *);
extern void clearLLD(LLD);
extern LLD copyLegs(LLD);
extern LLD copyLeg(LLD);
extern void setCurrentToFirst(LLD);
extern void setCurrentToLast(LLD);
extern void setCurrentToNext(LLD);
extern void setCurrentToPrev(LLD);
extern void setCurrentToPosition(LLD, void *);
extern BOOLEANS isCurrentValid(LLD);
extern void *getOrderData(LLD, int);
extern void *getCurrentData(LLD);
extern LLD getReverseList(ptr_net_t, LLD);

```

```
extern void setCurrentData(LLD, void *);
extern void insertFirst(LLD, void *);
extern void insertLast(LLD, void *);
extern void insertAfterCurrent(LLD, void *);
extern void *deleteCurrent(LLD);
extern void deleteCurrentList(LLD);
extern LLD merge2LLD(LLD, LLD, void *, void *);
extern LLD Find_Min_Leg(LLD, LLD);
extern LLD Find_Min_Legs(LLD, LLD);
extern int sizeLLD(LLD);
extern BOOLEANS isEmptyLLD(LLD);

#endif /* __LLD_H */
```

```

/*-----*/
/*
/* TITLE:
/*     netheap.h
/*
/*
/* DESCRIPTION:
/*     A network heap header file
/*
/*
/* ORIGIN:
/*     Programmer: Xiaohong Xin
/*     Date:      20 February 1998
/*     Platform:  Windows 95
/*     Compiler:  Borland C++ 5.0
/*
/*-----*/
#ifndef _heap_h
#define _heap_h

#ifndef _BOOL_
#define _BOOL_
typedef enum {false=0, true} BOOLEANS;
#endif

typedef struct pair
{
    int vertex;
    float priority;
} pair_t, *ptr_pair_t;

typedef struct heapentry
{
    ptr_pair_t p;
    int backpointer;
} heapentry_t, *ptr_heapentry_t;

typedef struct heap
{
    int *dataloc;
    heapentry_t *hpairs;
    int num;
    int max;
    int insertNum;
    int current;
} minheap_t, *ptr_minheap_t;

extern ptr_minheap_t createHeap(int);
extern void destroyHeap(minheap_t *);
extern void clearHeap(minheap_t *);
extern BOOLEANS isEmptyHeap(minheap_t *);
extern BOOLEANS areInsertionsAllowed(minheap_t *);
extern void * insertHeap(minheap_t *, ptr_pair_t);
extern void * addVertexToHeap(minheap_t *, int, float);
extern ptr_pair_t heapDeleteMin(minheap_t *);
extern void heapDecreaseKey(minheap_t *, int, float);
extern ptr_pair_t firstHeapEntry(minheap_t *);
extern ptr_pair_t nextHeapEntry(minheap_t *);

#endif

```

```

/*-----*/
/*
/* TITLE:
/*     matching.h
/*
/*
/* DESCRIPTION:
/*     A matching header file
/*
/*
/* ORIGIN:
/*     Programmer: Xiaohong Xin
/*     Date:       20 February 1998
/*     Platform:  Windows 95
/*     Compiler:   Borland C++ 5.0
/*
/*-----*/
#ifndef _MATCHING_H
#define _MATCHING_H

#define MATCHED 220
#define REPORT_BLOSSOMS TRUE

#define if_end_quit(e, net) \
if (e->indexID==net->links->num-1) break

#define forall_edges(e,net) \
for(e=net->links->v[0];e<net->links->v[net->links->num-1];adj_succ_edge(e,net))

#define first_adj_edge(e, v, net) \
e = net->links->v[net->fstar->bnode[v]]

#define adj_succ_edge(e, net) \
e = (e->indexID >= net->links->num-1)? \
net->links->v[net->links->num-1]:net->links->v[e->indexID+1]
/*
#define forall_adj_edges(e, v, net) \
for(e=net->links->v[net->fstar->bnode[v]]; \
    e<net->links->v[net->fstar->bnode[v+1]-1]; \
    adj_succ_edge(e, net))
*/
#define last_adj_edge(lastE, v, net) \
lastE = net->links->v[net->fstar->bnode[v+1]-1]

#define forall_nodes(v, net) \
for(v=0;v<net->nodes->num;v++)

#define source(e, net) \
net->fstar->anode[net->links->v[e->indexID]->anode]

#define target(e, net) \
net->fstar->anode[net->links->v[e->indexID]->bnode]

#define bridge_edge(x, net, map) \
net->links->v[map->v[x]->bridge]

#define add_links_from_LLD(aSet, links) \
setCurrentToFirst(aSet); \
while(aSet->current!=NULL) \
{insertLast(links, aSet->current->data);\
 setCurrentToNext(aSet);} \
destroyLLD(&aSet)

enum LABEL {ODD, EVEN, UNREACHED};

typedef struct DISJOINT
{
    int father;
    int next;
    int size;
} disjoint_t, *ptr_disjoint_t;

typedef struct PSEUDONODE
{
    LLD nodes;
    LLD links;
    int mu; /*dual variables for blossoms */
} pseudonode_t, *ptr_pseudonode_t;

```



```

typedef struct DUAL
{
    Queue * tao;
    float alpha;
}dual_t, *ptr_dual_t;

typedef struct MATCHNODE
{
    int nodeID;
    int label;
    int pred;
    int mate;
    int bridge;
    int all_matched;
    int path1;
    int path2;
    int passFlag;
    dual_t aDual;
    pseudonode_t aPseudo;
    disjoint_t aSet;
} matchnode_t, *ptr_matchnode_t;

typedef struct MATCHNODES
{
    int max;
    int size;
    int num;
    matchnode_t ** v;
}matchnodes_t, *ptr_matchnodes_t;

extern int Compare_Node_Degree(const void *, const void *);
extern void init_adj_iterator(int, ptr_net_t, ptr_matchnodes_t);
extern ptr_matchnodes_t Init_Matching(ptr_net_t);
extern ptr_matchnode_t PseudoMapNode(int,int, LLD, LLD, int, ptr_matchnodes_t);
extern void ShrinkNet(int, ptr_matchnode_t, ptr_net_t);
extern void UnShrinkNet(int, ptr_matchnodes_t map, ptr_net_t);
extern void min_node_e(ptr_link_t, int, ptr_net_t, ptr_matchnodes_t, ptr_minheap_t);
extern ptr_pair_t min_alpha(int, ptr_net_t);
extern ptr_pair_t min_e(int, int, ptr_net_t, ptr_matchnodes_t);
extern ptr_minheap_t min_even_shrink(int, ptr_net_t, ptr_matchnodes_t);
extern void beta_alpha(Queue *, float, float, ptr_link_t, ptr_net_t, ptr_matchnodes_t);
extern void UpdateEvenDual(int, ptr_matchnodes_t, ptr_net_t);
extern void UpdateShrinkDual(int, ptr_matchnodes_t, ptr_net_t);
extern void UpdateUnshrinkDual(int, ptr_matchnodes_t, ptr_net_t);
extern void DualChange(ptr_matchnodes_t);
extern BOOLEANs e_exist(ptr_link_t, ptr_matchnodes_t, ptr_net_t);
extern void Init_Base(ptr_matchnodes_t, ptr_net_t);
extern int find(int, ptr_matchnodes_t);
extern void union_blocks(int, int, ptr_matchnodes_t);
extern void make_rep(int, ptr_matchnodes_t);
extern int NODEID(int, ptr_net_t);
extern int LINKID(ptr_link_t);
extern LLD NODELINKS(int, int, ptr_net_t);
extern int greedy(ptr_net_t, ptr_matchnodes_t);
extern void heuristic(ptr_net_t, ptr_matchnodes_t, ptr_minheap_t);
extern void find_path(LLD, ptr_matchnodes_t, int, int, ptr_net_t);
extern LLD max_card_matching(ptr_net_t, int);
extern void Write_Matching(FILE *, LLD);

#endif

```

```

/*-----*/
/*
/* TITLE:
/*   matching.c(non bipartite minimum cost perfect matching problem library)*/
/*
/* DESCRIPTION:
/*   A library of an implementation of Edmonds' and Murty's algorithm
/*   with an application of the disjoint sets described by Tarjan
/*
/* FUNCTION NAMES:
/*   Functions for Implementing Disjoint Sets
/*   -----
/*
/*   1. void Init_Base(ptr_matchnodes_t, ptr_net_t)
/*   2. int Find(int, ptr_matchnodes_t)
/*   3. void union_blocks(int, int, ptr_matchnodes_t)
/*   4. void make_rep(int, ptr_matchnodes_t)
/*
/*   Functions for Manipulating a Matching Network
/*   -----
/*
/*   5. int NODEID(int, ptr_net_t)
/*   6. int LINKID(ptr_link_t)
/*   7. LLD NODELINKS(int, int, ptr_net_t)
/*   8. int Compare_Node_Degree(const void *, const void *)
/*   9. void init_adj_iterator(int, ptr_net_t, ptr_matchnodes_t)
/*   10. void min_node_e(ptr_link_t, int, ptr_net_t, ptr_matchnodes_t,
/*       ptr_minheap_t)
/*   11. ptr_pair_t min_e(int, int, ptr_net_t, ptr_matchnodes_t)
/*   12. BOOLEANs e_exist(ptr_link_t, ptr_matchnodes_t, ptr_net_t)
/*   13. void iterate_e(ptr_matchnodes_t, ptr_net_t)
/*
/*   Functions for primal-dual blossom matching
/*   -----
/*
/*   14. ptr_matchnodes_t Init_Matching(ptr_net_t)
/*   15. int greedy(ptr_net_t, ptr_matchnodes_t)
/*   16. void heuristic(ptr_net_t, ptr_matchnodes_t, ptr_minheap_t)
/*   17. void find_path(LLD, ptr_matchnodes_t, int, int, ptr_net_t)
/*   18. LLD max_card_matching(ptr_net_t, int)
/*   19. ptr_matchnode_t PseudoMapNode(int, int, LLD, LLD, int,
/*       ptr_matchnodes_t)
/*   20. void ShrinkNet(int, ptr_matchnode_t, ptr_net_t)
/*   21. void UnShrinkNet(int, ptr_matchnodes_t, ptr_net_t)
/*   22. ptr_pair_t min_alpha(int, ptr_net_t)
/*   23. ptr_minheap_t min_even_shrink(int, ptr_net_t, ptr_matchnodes_t)
/*   24. void beta_alpha(Queue *, float, float, ptr_link_t, ptr_net_t,
/*       ptr_matchnodes_t)
/*   25. void UpdateEvenDual(int, ptr_matchnodes_t, ptr_net_t)
/*   26. void UpdateShrinkDual(int, ptr_matchnodes_t, ptr_net_t)
/*   27. void UpdateUnshrinkDual(int, ptr_matchnodes_t, ptr_net_t)
/*   28. void DualChange(ptr_matchnodes_t)
/*   29. void Write_Matching(FILE *, LLD)
/*
/* REFERENCES:
/*   J. Edmonds: Paths, trees, and flowers
/*   Canad. J. Math., Vol. 17, 1965, 449-467
/*
/*   R.E. Tarjan: Data Structures and Network Algorithms,
/*   CBMS-NFS Regional Conference Series in Applied Mathematics,
/*   Vol. 44, 1983
/*
/*   Katta G. Murty: Network Programming
/*   Prentice Hall, Englewood Cliffs, New Jersey 07632
/*
/* RUNNING TIME:
/*   O(n*n*n)
/*
/* ORIGIN:
/*   Programmer: Xiaohong Xin
/*   Date: 20 February 1998
/*   Platform: Windows 95
/*   Compiler: Borland C++ 5.0
/*-----*/

```

```

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <memory.h>

```

```

#include "network.h"
#include "netlib.h"
#include "netheap.h"
#include "path.h"
#include "lld.h"
#include "buildtour.h"
#include "queue.h"
#include "matching.h"

/*****
/* Title:      Init_Base
/*
/* Description:
/*      Create new sets containing a single element.
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      map      A dynamic array of nodes in a matching network
/*      net      A network with a forward star data structure
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      void     No return value
*****/
void Init_Base(ptr_matchnodes_t map, ptr_net_t net)
{
    int x;

    for (x=0; x<net->nodes->num; x++)
    {
        map->v[x]->aSet.father = -1;
        map->v[x]->aSet.next = -1;
        map->v[x]->aSet.size = 1;
    }
}

/*****
/* Title:      find
/*
/* Description:
/*      Find the root node index for a disjoint set by changing the
/*      structure of a tree and moving closer to the root
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      y         The index for a disjoint set
/*      map      A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      root     The root index
*****/
int find(int y, ptr_matchnodes_t map)
{
    register int root;
    register int x;

    x = map->v[y]->aSet.father;
    if (x == -1) return y;
    root = y;
    while (map->v[root]->aSet.father != -1)
        root = map->v[root]->aSet.father;
    while (x != root)
    {
        map->v[y]->aSet.father = root;
        y = x;
        x = map->v[y]->aSet.father;
    }
    return root;
}

/*****
/* Title:      union_blocks
/*
/* Description:

```

```

/*      Form a new set that is the union of the two sets who contains      */
/*      elements a and b                                                  */
/*      */
/* Inputs:                                                                 */
/*      Name          Description                                         */
/*      -----      - - - - -                                           */
/*      a              one disjoint set element                          */
/*      b              another disjoint set element                       */
/*      map            A dynamic array of nodes in a matching network    */
/*      */
/* Outputs:                                                             */
/*      Name          Description                                         */
/*      -----      - - - - -                                           */
/*      void          No return value                                     */
/*      */
/*****
void union_blocks(int a, int b, ptr_matchnodes_t map)
{
    a = find_root(a, map);
    b = find_root(b, map);

    if (a == b)
        return;
    if (map->v[a]->aSet.size > map->v[b]->aSet.size)
    {
        map->v[b]->aSet.father = a;
        map->v[a]->aSet.size += map->v[b]->aSet.size;
    }
    else
    {
        map->v[a]->aSet.father = b;
        map->v[b]->aSet.size += map->v[a]->aSet.size;
    }
}

/*****
/* Title:      make_rep                                                  */
/*      */
/* Description:                                         */
/*      Make a node as a set if its index is equal to another root index. */
/*      */
/* Inputs:                                                                 */
/*      Name          Description                                         */
/*      -----      - - - - -                                           */
/*      base          A root index                                        */
/*      map            A dynamic array of nodes in a matching network    */
/*      */
/* Outputs:                                                             */
/*      Name          Description                                         */
/*      -----      - - - - -                                           */
/*      void          No return value                                     */
/*      */
/*****
void make_rep(int base, ptr_matchnodes_t map)
{
    if (Find_Root(base, map) == base)
        map->v[base]->aSet.father = -1;
}

/*****
/* Title:      Compare_Node_Degree                                       */
/*      */
/* Description:                                         */
/*      Compare nodes degrees                                           */
/*      */
/* Inputs:                                                                 */
/*      Name          Description                                         */
/*      -----      - - - - -                                           */
/*      arg1          Node ID                                             */
/*      arg2          Node ID                                             */
/*      */
/* Outputs:                                                             */
/*      Name          Description                                         */
/*      -----      - - - - -                                           */
/*      int           comparison result as an integer                     */
/*      */
/*****
int Compare_Node_Degree(const void *arg1, const void *arg2)
{
    node_t *g1;
    node_t *g2;

    g1 = *(node_t **)arg1;

```

```

    g2 = *(node_t **)arg2;
    return (g1->undirectedLinks - g2->undirectedLinks);
}

/*****
/* Title:      init_adj_iterator
/*
/* Description:
/*      Initialize a node v's adjacent edges for iteration
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      v          A node index
/*      net        A network with a forward star data structure
/*      map        A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      void      No return value
*****/
void init_adj_iterator(int v, ptr_net_t net, ptr_matchnodes_t map)
{
    int i;
    ptr_link_t e;

    map->v[v]->passFlag = NOPASS;
    for(i=net->fstar->bnode[v];i<net->fstar->bnode[v+1];i++)
    {
        e=net->links->v[i];
        e->passFlag = NOPASS;
        ReverseLink(net, e)->passFlag = NOPASS;
    }
}

/*****
/* Title:      Init_Matching
/*
/* Description:
/*      Initialize a matching network.
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      net        A network with a forward star data structure
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      matchnodes nodes in a matching network
*****/
ptr_matchnodes_t Init_Matching(ptr_net_t net)
{
    int x;
    ptr_pair_t minPair;
    Queue * aQ;
    matchnode_t * matchnode;
    matchnodes_t * matchnodes;

    matchnodes = (matchnodes_t *) Init_Header(sizeof(matchnode_t));
    for (x=0; x<net->nodes->num; x++)
    {
        matchnode = (matchnode_t *)New Element(sizeof(matchnode_t));
        matchnode->nodeID = net->nodes->v[x]->nodeID;
        matchnode->label = INVALID;
        matchnode->pred = INVALID;
        matchnode->mate = INVALID;
        matchnode->bridge = INVALID;
        matchnode->all_matched = INVALID;
        matchnode->path1 = INVALID;
        matchnode->path2 = INVALID;
        matchnode->passFlag = PASS;
        minPair = min_alpha(x, net);

        /*initialize original node prices */
        matchnode->aDual.alpha = (minPair->priority)/2;
        aQ = MakeQueue(net->links->num);
        Enqueue(aQ, minPair->vertex);
        matchnode->aDual.tao = aQ; /* make a queue to store all */
    }
}

```

```

    matchnode->aPseudo.mu = 0;          /* initialize pseudonode prices */
    matchnode->aPseudo.nodes = NULL; /* only for pseudo nodes */
    matchnode->aPseudo.links = NULL; /* only for pseudo nodes */
    Append_Element((header_t *)matchnodes, (void *)matchnode);
}
return matchnodes;
}

/*****
/* Title:      Pseudo_Map_Node
/*
/* Description:
/*      Create pseudonode and set its parameters
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      mMatch        The number of matching pairs
/*      count          offset of a new pseudo node index
/*      nodes          node list in a blossom
/*      links          link list in a blossom
/*      BaseNodeIndex pseudonode index
/*      map            A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      bnode         a pseudonode
*****/
ptr_matchnode_t Pseudo_Map_Node(int nMatch,int count, LLD nodes, LLD links, int BaseNodeIdx,
ptr_matchnodes_t map)
{
    ptr_matchnode_t bNode;
    ptr_matchnode_t BaseNode;
    Queue * aQ;

    aQ = MakeQueue(map->num);
    BaseNode = map->v[BaseNodeIdx];
    bNode = (matchnode_t *)New_Element(sizeof(matchnode_t));
    bNode->nodeID = map->v[map->num-1]->nodeID + count;
    bNode->label = BaseNode->label;
    bNode->pred = BaseNode->pred;
    bNode->mate = BaseNode->mate;
    bNode->bridge = BaseNode->bridge;
    bNode->all_matched = BaseNode->all_matched;
    bNode->path1 = BaseNode->path1;
    bNode->path2 = BaseNode->path2;
    bNode->passFlag = PSEUDO;
    bNode->aPseudo.nodes = nodes;
    bNode->aPseudo.links = links;
    if (nMatch == (sizeLLD(nodes)-1)/2)
        bNode->aPseudo.mu = 1;
    else
        bNode->aPseudo.mu = 0;
    bNode->aDual.alpha = BaseNode->aDual.alpha;
    bNode->aDual.tao = BaseNode->aDual.tao;
    bNode->aSet.father = BaseNode->aSet.father;
    bNode->aSet.next = BaseNode->aSet.father;
    bNode->aSet.size = BaseNode->aSet.size;
    Append_Element((header_t *)map, (void *)bNode);
    return bNode;
}

/*****
/* Title:      ShrinkNet
/*
/* Description:
/*      A new matching network with a pseudonode
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      count          offset from the root index
/*      pseudonodes    a pseudonode
/*      net            A network with a forward star data structure
/*
/* Outputs:
/*      Name          Description
*****/

```

```

/*      void                No return value                */
/*****
void ShrinkNet(int count, ptr_matchnode_t pseudonode, ptr_net_t net)
{
    node_t * xNode;
    ptr_matchnode_t aNode;
    ptr_link_t aLink, e, cLink;
    int v, xNodeID,i;
    LLD nodes;
    LLD links;

    nodes = pseudonode->aPseudo.nodes;
    links = pseudonode->aPseudo.links;

    /*make nodes in Blossom unpassed */
    setCurrentToFirst(nodes);
    while (nodes->current != NULL)
    {
        aNode = getCurrentData(nodes);
        /* printf("A node in nodes is %d \n", aNode->nodeID); */
        aNode->passFlag = BLOSSOM;
        setCurrentToNext(nodes);
    }

    /* make links in Blossom unpassed */
    setCurrentToFirst(links);
    while (links->current != NULL)
    {
        aLink = getCurrentData(links);
        aLink->passFlag = BLOSSOM;
        setCurrentToNext(links);
    }

    /*add a pseudonode with the highest node id to net */
    xNodeID = net->nodes->v[net->nodes->num-1]->nodeID+count;
    xNode = New_Node(xNodeID);
    Append_Element ((header_t *)net->nodes, (void *)xNode);

    /* add pseudolinks to net */
    setCurrentToFirst(nodes);
    while(nodes->current != NULL)
    {
        aNode = getCurrentData(nodes);
        v = net->fstar->anode[aNode->nodeID];
        for (i=net->fstar->bnode[v];i<net->fstar->bnode[v+1];i++)
        {
            e = net->links->v[i];
            if (e->passFlag != BLOSSOM)
            {
                cLink = CopyLink(e);
                cLink->anode = xNodeID;
                cLink->passFlag = PASS;
                Append_Element ((header_t *)net->links, (void *)cLink);
                cLink = ReverseLink(net, CopyLink(e));
                cLink = CopyLink(cLink);
                cLink->bnode = xNodeID;
                cLink->passFlag = PASS;
                Append_Element ((header_t *)net->links, (void *)cLink);
            }
        }
        setCurrentToNext(nodes);
    }
    free(net->fstar->anode);
    free(net->fstar->bnode);
    Build_Forward_Star (net);
}

/*****
/* Title:                UnSrinkNet                */
/*                      */
/* Description:          */
/*      Unshrinking from a pseudonode                */
/*                      */
/* Inputs:              */
/*      Name                Description                */
/*      -----                -----                */
/*      v                a pseudonode index                */
/*      map                A dynamic array of nodes in a matching network                */
/*      net                A network with a forward star data structure                */
/*****/

```

```

/* Outputs:                                                                    */
/*   Name           Description                                                */
/* -----          - - - - -                                                */
/*   void           No return value                                           */
/*****
void UnShrinkNet(int v, ptr_matchnodes_t map, ptr_net_t net)
{
    int i;
    ptr_matchnode_t aNode;
    ptr_link_t      aLink, e;
    LLD nodes;
    LLD links;

    nodes = map->v[v]->aPseudo.nodes;
    links = map->v[v]->aPseudo.links;

    /* Since psIdx has already been with PSEUDO passFlag, the edge for it has */
    /* to be changed to PSEUDO in case not to be traversed                    */
    for(i=net->fstar->bnode[v];i<net->fstar->bnode[v+1];i++)
    {
        e = net->links->v[i];
        e->passFlag = PSEUDO;
        ReverseLink(net, e)->passFlag = PSEUDO;
    }

    setCurrentToFirst(nodes);
    while(nodes->current != NULL)
    {
        aNode = getCurrentData(nodes);
        aNode->passFlag = PASS;
        setCurrentToNext(nodes);
    }

    setCurrentToFirst(links);
    while(links->current != NULL)
    {
        aLink = getCurrentData(links);
        aLink->passFlag = PASS;
        setCurrentToNext(links);
    }
}

/*****
/* Title:      NODEID                                                         */
/*                                                    */
/* Description:                                                         */
/*   Get user id for a node                                             */
/*                                                    */
/* Inputs:                                                         */
/*   Name           Description                                           */
/* -----          - - - - -                                           */
/*   v               a node index                                         */
/*   net             A network with a forward star data structure        */
/*                                                    */
/* Outputs:                                                         */
/*   Name           Description                                           */
/* -----          - - - - -                                           */
/*   out            A node user id                                         */
/*****
int NODEID(int v, ptr_net_t net)
{
    int out;

    out = net->nodes->v[v]->nodeID;
    return out;
}

/*****
/* Title:      LINKID                                                         */
/*                                                    */
/* Description:                                                         */
/*   Get user id for an edge                                             */
/*                                                    */
/* Inputs:                                                         */
/*   Name           Description                                           */
/* -----          - - - - -                                           */
/*   e               an edge index                                         */
/*                                                    */
/* Outputs:                                                         */
/*   Name           Description                                           */

```



```

/* ----- */
/* out An edge user id */
/******/
int LINKID(ptr_link_t e)
{
    int out;

    out = e->arc_id;
    return out;
}

/******/
/* Title: NODELINKS */
/* Description: */
/* Get user id for a node */
/* Inputs: */
/* Name Description */
/* ----- */
/* v a node index */
/* net A network with a forward star data structure */
/* Outputs: */
/* Name Description */
/* ----- */
/* out A node user id */
/******/
LLD NODELINKS(int v, int w, ptr_net_t net)
{
    int i;
    LLD linkSet;
    ptr_link_t e;

    linkSet = createLLD();
    for(i=net->fstar->bnode[v];i<net->fstar->bnode[v+1];i++)
    {
        e = net->links->v[i];
        if(net->fstar->anode[e->bnode]==w)
        {
            insertLast(linkSet, e);
            insertLast(linkSet, ReverseLink(net,e));
        }
    }
    return linkSet;
}

/******/
/* Title: min_node_e */
/* Description: */
/* find an unmatched incident edge from a node with minimum length */
/* Inputs: */
/* Name Description */
/* ----- */
/* v a node index */
/* net A network with a forward star data structure */
/* map A dynamic array of nodes in a matching network */
/* theHeap A heap for unmatched incident edges */
/* Outputs: */
/* Name Description */
/* ----- */
/* void No return value */
/******/
void min_node_e(int v, ptr_net_t net, ptr_matchnodes_t map, ptr_minheap_t theHeap)
{
    int i;

    clearHeap(theHeap);
    for(i=net->fstar->bnode[v];i<net->fstar->bnode[v+1];i++)
    {
        e = net->links->v[i];
        if (map->v[target(e,net)]->mate==INVALID&&e->passFlag==PASS)
            addVertexToHeap(theHeap, target(e,net), e->distance);
    }
}

/******/

```

```

/* Title:      min_alpha
/*
/* Description:
/*      find an incident edge from a node with minimum length
/*
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      v             a node index
/*      net           A network with a forward star data structure
/*
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      aPair        an edge with minimum length
/*****
ptr_pair_t min_alpha(int v, ptr_net_t net)
{
    int i;
    ptr_pair_t aPair;
    ptr_link_t e, rLink;
    ptr_minheap_t theHeap;

    theHeap = createHeap(1000);
    for (i=net->fstar->bnode[v]; i<net->fstar->bnode[v+1]; i++)
    {
        e=net->links->v[i];
        rLink = ReverseLink(net, e);
        addVertexToHeap(theHeap, rLink->indexID, rLink->distance);
    }
    aPair = heapDeleteMin(theHeap);
    destroyHeap(theHeap);
    return aPair;
}

/*****
/* Title:      min_e
/*
/* Description:
/*      find an unmatched incident edge from a node with minimum dual
/*
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      v             a node index
/*      pseudo        a pseudo node index
/*      net           A network with a forward star data structure
/*      map           A dynamic array of nodes in a matching network
/*
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      aPair        an edge with minimum dual variable
/*****
ptr_pair_t min_e(int v, int pseudo, ptr_net_t net, ptr_matchnodes_t map)
{
    int i;
    float aDual;
    ptr_pair_t aPair;
    ptr_link_t e, rLink;
    ptr_minheap_t theHeap = createHeap(net->nodes->num);

    theHeap = createHeap(net->nodes->num);
    for (i=net->fstar->bnode[v]; i<net->fstar->bnode[v+1]; i++)
    {
        e = net->links->v[i];
        rLink = ReverseLink(net, e);
        if (rLink->passFlag == BLOSSOM)
            aDual = rLink->distance-(map->v[v]->aDual.alpha+map->v[target(e, net)]->aDual.alpha-map-
>v[pseudo]->aPseudo.mu);
        else
            aDual = rLink->distance-(map->v[v]->aDual.alpha+map->v[target(e, net)]->aDual.alpha);
        addVertexToHeap(theHeap, rLink->indexID, aDual);
    }
    aPair = heapDeleteMin(theHeap);
    destroyHeap(theHeap);
    return aPair;
}

/*****

```

```

/* Title:      min_even_shrink
/*
/* Description:
/*      build a dual heap over even current nodes and blossom nodes
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      v             a node index
/*      net           A network with a forward star data structure
/*      map           A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      theHeap      an output heap
/*****
ptr_minheap_t min_even_shrink(int v, ptr_net_t net, ptr_matchnodes_t map)
{
    int i;
    float aDual;
    ptr_link_t e, rLink;
    ptr_minheap_t theHeap;

    theHeap = createHeap(net->nodes->num);
    for(i=net->fstar->bnode[v];i<net->fstar->bnode[v+1];i++)
    {
        e = net->links->v[i];
        rLink = ReverseLink(net, e);
        if (rLink->passFlag != BLOSSOM && map->v[net->fstar->anode[rLink->anode]]->label==EVEN)
        {
            aDual = rLink->distance-(map->v[v]->aDual.alpha+map->v[target(e,net)]->aDual.alpha);
            addVertexToHeap(theHeap, rLink->indexID, aDual);
        }
    }
    return theHeap;
}

/*****/
/* Title:      beta_alpha
/*
/* Description:
/*      build a dual heap over even current nodes and blossom nodes
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      aQ            A Queue
/*      beta          beta
/*      f             another beta for comparison
/*      net           A network with a forward star data structure
/*      map           A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
/*****/
void beta_alpha(Queue * aQ, float beta, float j, ptr_link_t e, ptr_net_t net, ptr_matchnodes_t map)
{
    int aLink;

    if (beta == j)
    {
        if (!QueueEmpty(aQ))
            aLink = Dequeue(aQ);
        else
            return;
        if(aLink != e->indexID)
            Enqueue(aQ, e->indexID);
        Enqueue(aQ, aLink);
    }

    if (beta < j)
    {
        InitQueue(aQ);
        Enqueue(aQ, e->indexID);
        map->v[net->fstar->anode[e->bnode]]->aDual.alpha = beta;
    }
}

```

```

/*****/
/* Title:      update_even_dual */
/* */
/* Description: */
/*      Update when an unlabeled node is labeled as an even node */
/* */
/* Inputs: */
/*      Name      Description */
/*      -----      ----- */
/*      v          a node index */
/*      net        A network with a forward star data structure */
/*      map        A dynamic array of nodes in a matching network */
/* */
/* Outputs: */
/*      Name      Description */
/*      -----      ----- */
/*      void      No return value */
/*****/

```

```

void update_even_dual(int v, ptr_matchnodes_t map, ptr_net_t net)
{
    int k;
    float beta, j, i;
    ptr_link_t e;
    Queue * aQ;

    for(k=net->fstar->bnode[v];k<net->fstar->bnode[v+1];k++)
    {
        e = net->links->v[k];
        aQ = map->v[net->fstar->anode[e->bnode]]->aDual.tao;
        i = map->v[v]->aDual.alpha;
        j = map->v[net->fstar->anode[e->bnode]]->aDual.alpha;
        beta = e->distance - (i + j);
        beta_alpha(aQ, beta, j, e, net, map);
    }
}

```

```

/*****/
/* Title:      update_shrink_dual */
/* */
/* Description: */
/*      Update when a blossom is shrunken */
/* */
/* Inputs: */
/*      Name      Description */
/*      -----      ----- */
/*      pIdx      a pseudonode index */
/*      net        A network with a forward star data structure */
/*      map        A dynamic array of nodes in a matching network */
/* */
/* Outputs: */
/*      Name      Description */
/*      -----      ----- */
/*      void      No return value */
/*****/

```

```

void update_shrink_dual(int pIdx, ptr_matchnodes_t map, ptr_net_t net)
{
    int v, i;
    Queue *aQ, * aLinkQ;
    ptr_minheap_t aHeap;
    ptr_minheap_t theHeap = createHeap(net->links->num);
    ptr_minheap_t elseHeap = createHeap(net->links->num);
    ptr_pair_t aPair;
    ptr_link_t e, oneMinLink;
    float d, beta, j;
    ptr_matchnode_t aNode;

    aLinkQ = MakeQueue(map->num);
    theHeap = createHeap(net->links->num);
    elseHeap = createHeap(net->links->num);

    /* compute min among all the edges of blossom nodes */
    LLD nodes = map->v[pIdx]->aPseudo.nodes;
    setCurrentToFirst(nodes);
    while (nodes->current != NULL)
    {
        aNode = getCurrentData(nodes);
        v = net->fstar->anode[aNode->nodeID];
    }
}

```

```

aPair = min_e(v, pIdx, net, map); /*find the total min */
addVertexToHeap(theHeap, aPair->vertex, aPair->priority);

aHeap = min_even_shrink(v, net, map); /* find the even min */
aPair = heapDeleteMin(aHeap);
d = aPair->priority;
do
{
    oneMinLink = net->links->v[aPair->vertex];
    addVertexToHeap(elseHeap, oneMinLink->indexID, oneMinLink->distance);
    aPair = heapDeleteMin(aHeap);
} while (!isEmptyHeap(aHeap) && d==aPair->priority);

/* make a queue for links outside the blossom */
for(i=net->fstar->bnode[v]; i<net->fstar->bnode[v+1]; i++)
{
    e = net->links->v[i];
    if (e->passFlag == BLOSSOM)
        continue;
    Enqueue(aLinkQ, e->indexID);
}
setCurrentToNext(nodes);
beta = heapDeleteMin(theHeap)->priority;

/* compare alpha with beta for every link linking to the blossom */
while(!QueueEmpty(aLinkQ) )
{
    e = net->links->v[Dequeue(aLinkQ)];
    aQ = map->v[net->fstar->anode[e->anode]]->aDual.tao;
    j = map->v[net->fstar->anode[e->anode]]->aDual.alpha;
    beta_alpha(aQ, beta, j, e, net, map);
}

/* replace alpha for the pseudo node. elseHeap elements have the same */
/* priority */
while (!isEmptyHeap(elseHeap))
{
    aPair = heapDeleteMin(elseHeap);
    Enqueue(map->v[pIdx]->aDual.tao, aPair->vertex);
}
map->v[pIdx]->aDual.alpha = aPair->priority;
destroyHeap(aHeap);
destroyHeap(theHeap);
destroyHeap(elseHeap);
}

```

```

/*****
/* Title:      update_unshrink_dual */
/* */
/* Description: */
/* Update when a blossom is unshrunk */
/* */
/* Inputs: */
/* Name      Description */
/* ----- */
/* pIdx      a pseudonode index */
/* net       A network with a forward star data structure */
/* map       A dynamic array of nodes in a matching network */
/* */
/* Outputs: */
/* Name      Description */
/* ----- */
/* void     No return value */
*****/

```

```

void update_unshrink_dual(int pIdx, ptr_matchnodes_t map, ptr_net_t net)
{

```

```

    ptr_matchnode_t m;
    int v, i;
    float beta, j;
    Queue * aQ;
    LLD newLLD;
    ptr_minheap_t aHeap;
    ptr_link_t e;
    ptr_pair_t mPair;
    LLD nodes;

    newLLD = createLLD();
    aHeap = createHeap(map->num);

```

```

nodes = map->v[pIdex]->aPseudo.nodes;
setCurrentToFirst(nodes);
while (nodes->current != NULL)
{
    m = getCurrentData(nodes);
    v = net->fstar->anode[m->nodeID];

    for(i=net->fstar->bnode[v]; i<net->fstar->bnode[v+1]; i++)
    {
        e = net->links->v[i];
        if (e->passFlag != BLOSSOM)
            insertLast(newLLD, ReverseLink(net,e));
    }

    if (map->v[v]->label==EVEN)
    {
        mPair = min_e(v, pIdex, net, map);
        addVertexToHeap(aHeap, mPair->vertex, mPair->priority);
    }
    setCurrentToNext(nodes);
}
beta = heapDeleteMin(aHeap)->priority;

setCurrentToFirst(newLLD);
while (newLLD->current != NULL)
{
    e = getCurrentData(newLLD);
    v = net->fstar->anode[e->anode];
    aQ = map->v[v]->aDual.tao;
    j = map->v[v]->aDual.alpha;
    beta_alpha(aQ, beta, j, e, net, map);
    setCurrentToNext(newLLD);
}
destroyHeap(aHeap);
destroyLLD(&newLLD);
}

/*****
/* Title:      dual_change
/*
/* Description:
/*      carry out a dual solution change, mainly compute the alpha variable
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      map      A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      void      No return value
*****/
void dual_change(ptr_matchnodes_t map)
{
    int i;
    float minalpha;
    ptr_minheap_t unheap;
    ptr_minheap_t evenheap;
    ptr_minheap_t pseudoheap;
    ptr_minheap_t aheap;

    unheap = createHeap(map->num);
    evenheap = createHeap(map->num);
    pseudoheap = createHeap(map->num/2);
    aheap = createHeap(map->num);

    /*pretty akward ways to find the minimum alpha, but it works! */
    for (i = 0; i < map->num; i++)
    {
        if (map->v[i]->label == INVALID)
        {
            addVertexToHeap(unheap, i, map->v[i]->aDual.alpha);
            if (map->v[i]->label == EVEN)
                addVertexToHeap(evenheap, i, map->v[i]->aDual.alpha);
            if (map->v[i]->passFlag == PSEUDO)
                addVertexToHeap(pseudoheap, i, map->v[i]->aPseudo.mu);
        }
    }
    if (!isEmptyHeap(unheap))

```

```

        addVertexToHeap(aheap, 1, heapDeleteMin(unheap)->priority);
    if (!isEmptyHeap(evenheap))
        addVertexToHeap(aheap, 2, (heapDeleteMin(evenheap)->priority)/2);
    if (!isEmptyHeap(pseudoheap))
        addVertexToHeap(aheap, 3, (heapDeleteMin(pseudoheap)->priority)/2);
    if (!isEmptyHeap(aheap))
        minalpha = heapDeleteMin(aheap)->priority;
    else
    {
        printf("Warn: The dual heap is empty, something is wrong.");
        exit(1);
    }
    if (minalpha > INFINITY)
        return;
    if (minalpha != 0.0)
    {
        for (i = 0; i < map->num; i++)
        {
            if (map->v[i]->label == ODD)
                map->v[i]->aDual.alpha -= minalpha;
            if (map->v[i]->label == EVEN)
                map->v[i]->aDual.alpha += minalpha;
            if (map->v[i]->passFlag == PSEUDO)
                map->v[i]->aPseudo.mu -= 2*minalpha;
        }
    }
    destroyHeap(unheap);
    destroyHeap(evenheap);
    destroyHeap(pseudoheap);
    destroyHeap(aheap);
    return;
}

/*****
/* Title:      e_exist
/*
/* Description:
/*      Decide an edge with index e belonging to the net or not
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      e          an edge index
/*      net        A network with a forward star data structure
/*      map        A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      BOOLEANs  a boolean value
*****/
BOOLEANS e_exist(ptr_link_t e, ptr_matchnodes_t map, ptr_net_t net)
{
    int firstLink, aLink;
    Queue * aQ;

    aQ = map->v[net->fstar->anode[e->anode]]->aDual. tao;
    firstLink = Dequeue(aQ);
    aLink = firstLink;
    Enqueue(aQ, firstLink);
    do
    {
        if (net->links->v[aLink]->arc_id == e->arc_id)
            return true;
        aLink = Dequeue(aQ);
        Enqueue(aQ, aLink);
    } while (aLink != firstLink);

    return false;
}

/*****
/* Title:      iterate_e
/*
/* Description:
/*      Iterate the map's dual
/*
/* Inputs:
/*      Name      Description
/*      -----
*****/

```

```

/*      net          A network with a forward star data structure */
/*      map          A dynamic array of nodes in a matching network */
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
/*****
void iterate_e(ptr_matchnodes_t map, ptr_net_t net)
{
    int i, aLink;
    Queue * aQ;
    Queue * cQ;

    for (i = 0; i<map->num;i++)
    {
        Queue * aQ = map->v[i]->aDual.tao;
        Queue * cQ = MakeQueue(map->num);
        while (!QueueEmpty(aQ))
        {
            aLink = Dequeue(aQ);
            Enqueue(cQ, aLink);
        }
        map->v[i]->aDual.tao = cQ;
        DestroyQueue(aQ);
    }
}

/*****
/* Title:      greedy
/*
/* Description:
/*      greedy algorithm
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      net          A network with a forward star data structure
/*      map          A dynamic array of nodes in a matching network
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      count        the number of matching nodes
/*****
int greedy(ptr_net_t net, ptr_matchnodes_t map)
{
    int i, v, w, count;
    minheap_t * theHeap;
    ptr_pair_t aPair;

    count = 0;
    if((theHeap=createHeap(net->links->num))==NULL)
    {
        printf("Error:Can not create new heap.\n");
        exit(1);
    }

    for (i=0;i<net->links->num;i++)
        if(areInsertionsAllowed(theHeap))
            addVertexToHeap(theHeap, i, net->links->v[i]->distance);

    while (!isEmptyHeap(theHeap))
    {
        aPair = heapDeleteMin(theHeap);
        v=net->fstar->anode[net->links->v[aPair->vertex]->anode];
        w=net->fstar->anode[net->links->v[aPair->vertex]->bnode];
        if (map->v[v]->mate==INVALID&&map->v[w]->mate==INVALID)
        {
            map->v[v]->mate = w;
            map->v[w]->mate = v;
            count++;
        }
    }
    destroyHeap(theHeap);
    return count;
}

/*****
/* Title:      heuristic
/*

```



```

/*
/* Description:
/* (Markus Paul):
/* finds almost all augmenting paths of length <=3 with two passes over
/* the adjacency lists
/* ("almost": discovery of a blossom {v,w,x,v} leads to a skip of the
/* edge {x,v}, even if the base v stays unmatched - it's not worth while*/
/* to fix this problem)
/* if all adjacent nodes w of v are matched, try to find an other
/* partner for mate, and match v and w on success
/*
/* Inputs:
/* Name Description
/* -----
/* net A network with a forward star data structure
/* map A dynamic array of nodes in a matching network
/* theHeap A heap of incident edges for a node
/*
/* Outputs:
/* Name Description
/* -----
/* void No return value
/*
/*****
void heuristic(ptr_net_t net, ptr_matchnodes_t map, ptr_minheap_t theHeap)
{
    int u, v, w, x, i;
    ptr_link_t e = NULL;
    ptr_link_t f;
    ptr_pair_t aPair;

    forall_nodes(v, net)
    {
        if(map->v[v]->passFlag==NOPASS) continue;
        if(map->v[v]->mate==INVALID)
        {
            min_node_e(e, v, net, map, theHeap);
            if(!isEmptyHeap(theHeap))
            {
                aPair = heapDeleteMin(theHeap);
                map->v[v]->mate = aPair->vertex;
                map->v[aPair->vertex]->mate = v ;
            }
            else /* second pass */
            {
                map->v[v]->all_matched = MATCHED;
                for(i=net->fstar->bnode[v];i<net->fstar->bnode[v+1];i++)
                {
                    e = net->links->v[i];
                    w = target(e, net);
                    x = map->v[w]->mate;
                    if(map->v[x]->all_matched == INVALID)
                    {
                        int found = FALSE;
                        min_node_e(f, x, net, map, theHeap);
                        while(!isEmptyHeap(theHeap))
                        {
                            aPair = heapDeleteMin(theHeap);
                            u = aPair->vertex;

                            if(u != v && map->v[u]->mate==INVALID)
                            {
                                found = TRUE;
                                break;
                            }
                        }
                        clearHeap(theHeap);
                        if(found)
                        {
                            map->v[u]->mate = x;
                            map->v[x]->mate = u;
                            map->v[v]->mate = w;
                            map->v[w]->mate = v;
                            break;
                        }
                        else
                            map->v[x]->all_matched = MATCHED;
                    }
                }
            }
        }
    }
}

```

```

)
)

/*****
/* Title:      find_path
/*
/* Description:
/* computes an even length alternating path from x to y beginning with a
/* matching edge (Tarjan: Data Structures and Network Algorithms, page 122)
/* Preconditions:
/* a) x and y are even or shrunked
/* b) when x was made part of a blossom for the first time, y was a base
/* and predecessor of the base of that blossom
/*
/* Inputs:
/* Name          Description
/* -----
/* L              A doubly linked list to store x
/* net            A network with a forward star data structure
/* x              A node index
/* y              A node index
/* map            A dynamic array of nodes in a matching network
/*
/* Outputs:
/* Name          Description
/* -----
/* void          No return value
*****/
void find_path(LLD L, ptr_matchnodes_t map, int x, int y, ptr_net_t net)
{
    if (x==y)
    {
        insertLast(L, (int *)x);
        return;
    }

    if (map->v[x]->label == EVEN)
    {
        find_path(L,map,map->v[map->v[x]->mate]->pred,y, net);
        insertFirst(L, (int *) (map->v[x]->mate));
        insertFirst(L, (int *)x);
        return;
    }

    if (map->v[x]->label == ODD)
    {
        int v;
        LLD L1 = createLLD();
        Find_Path(L,map,target(bridge_edge(x, net, map), net),y, net);
        Find_Path(L1,map, source(bridge_edge(x, net, map), net), map->v[x]->mate, net);

        setCurrentToFirst(L1);
        while (L1->current != NULL)
        {
            insertFirst(L, getCurrentData(L1));
            setCurrentToNext(L1);
        }
        insertFirst(L, (int *)x);
        return;
    }
}

/*****
/* Title:      max_card_matching
/*
/* Description:
/* Primal_dual Matching algorithm
/*
/* Inputs:
/* Name          Description
/* -----
/* net            A network with a forward star data structure
/* heur           greedy or heuristic
/*
/* Outputs:
/* Name          Description
/* -----
/* LLD           A doubly linked list of matching edges
*****/

```

```

/*****
LLD max_card_matching(ptr_net_t net, int heur)
{
    LLD result;
    ptr_matchnodes_t map;
    int strue;
    BOOLEANS done;
    int a, b, v, i;
    ptr_link_t e;
    minheap_t * theHeap;

    strue = 0;
    done = false;
    if((theHeap=createHeap(net->nodes->num))==NULL)
    {
        printf("Error:Can not create new heap.\n");
        exit(1);
    }

    map = Init_Matching(net);
    iterate_e(map, net);
    switch (heur)
    {
        case 1:
        {
            greedy(net,map);
            break;
        }

        case 2:
        {
            heuristic(net, map, theHeap);
            break;
        }
    }
}

while (! done) /* main loop */
{
    Queue *Q = MakeQueue((net->nodes->num)*2);
    int nBlossom = 0, w;
    Init_Base(map, net);
    done = true;

    forall_nodes(v, net)
    {
        if (map->v[v]->passFlag==NOPASS||map->v[v]->passFlag==BLOSSOM)
            continue;
        if (map->v[v]->mate == INVALID)
        {
            map->v[v]->label = EVEN;
            w = net->nodes->v[v]->nodeID;
            Enqueue(Q, w);
        }
        else map->v[v]->label = UNREACHED;
    }

    /* search for augmenting path */
    while (!QueueEmpty(Q))
    {
        int w, i;
        int v = Dequeue(Q);
        ptr_link_t e;

        v = net->fstar->anode[v];
        if (map->v[v]->passFlag==NOPASS||map->v[v]->passFlag==BLOSSOM)
            continue;
        if (map->v[v]->passFlag==PSEUDO&&map->v[v]->aPseudo.mu==0)
            UpdateUnshrinkDual(v, map, net);
        for (i=net->fstar->bnode[v]; i<net->fstar->bnode[v+1]; i++)
        {
            e = net->links->v[i];
            if (e->passFlag==NOPASS||e->passFlag==BLOSSOM)
            {
                if_end_quit(e, net);
                continue;
            }
            w = target(e, net);

            if (v == w)
            {

```

```

        if_end_quit(e, net);
        continue; /* ignore self-loops */
    }

    if (find(v, map) == find(w, map) || (map->v[w]->label == ODD \
        && find(w, map) == w))
    {
        if_end_quit(e, net);
        continue; /* do nothing */
    }

    /*scan an outer node*/
    if (map->v[w]->label == UNREACHED&&map->v[v]->aDual.alpha > 0.0)
    {
        /* no change for labelling an odd node */
        map->v[w]->label = ODD;
        map->v[w]->pred = v;

        /*update when an unlabelled node as a even node*/
        map->v[map->v[w]->mate]->label = EVEN;
        UpdateEvenDual(map->v[w]->mate, map, net);
        Enqueue(Q, net->nodes->v[map->v[w]->mate]->nodeID);
    }
    else
    {
        int i;
        for (i=0;i<map->num;i++)
            iterate_e(map, net);
        if (e_exist(e, map, net))
        {
            int hv = find(v, map);
            int hw = find(w, map);
            strue++;
            map->v[hv]->path1 = map->v[hw]->path2 = strue;

            while ((map->v[hw]->path1 != strue && \
                map->v[hv]->path2 != strue)&&
                (map->v[hv]->mate != INVALID \
                || map->v[hw]->mate != INVALID) )
            {
                if (map->v[hv]->mate != INVALID)
                {
                    hv = find(map->v[map->v[hv]->mate]->pred, map);
                    map->v[hv]->path1 = strue;
                }

                if (map->v[hw]->mate != INVALID)
                {
                    hw = find(map->v[map->v[hw]->mate]->pred, map);
                    map->v[hw]->path2 = strue;
                }
            }

            /* Shrink Blossom */
            if (map->v[hw]->path1 == strue || \
                map->v[hv]->path2 == strue)
            {
                int x, y, nMatch=0;
                /* Base */
                int b = (map->v[hw]->path1==strue) ? hw : hv;
                ptr_matchnode_t pseudo;
                LLD aSet;
                LLD nodes = createLLD();
                LLD links = createLLD();

                aSet = NodeLinks(net->fstar->anode[e->anode], \
                    net->fstar->anode[e->bnode], net);
                add_links_from_LLD(aSet, links);

#ifdef REPORT_BLOSSOMS
                printf("SHRINK BLOSSOM\n");
                printf("bridge = %d\n", LINKID(e));
                printf("base   = %d\n", NODEID(b,net));
                printf("path1 = ");
#endif
                x = find(v, map);
                while (x != b)
                {
#ifdef REPORT_BLOSSOMS

```

```

    printf("%d ", NODEID(x,net));
#endif

    insertLast(nodes, map->v[x]);
    Union_Sets(x,b, map);
    Make_Replicate(b, map);

    aSet = NodeLinks(map->v[x]->mate, x, net);
    add_links_from_LLD(aSet, links);
    x = map->v[x]->mate;
    nMatch++;

#if defined(REPORT_BLOSSOMS)
    printf("%d ", NODEID(x,net));
#endif

    y = find(map->v[x]->pred, map);
    union_blocks(x, b, map);
    Make_Replicate(b, map);

    if (x != b)
    {
        insertLast(nodes, map->v[x]);
        aSet = NODELINKS(x,map->v[x]->pred, net);
        add_links_from_LLD(aSet, links);
    }
    Enqueue(Q, net->nodes->v[x]->nodeID);
    map->v[x]->bridge = e->indexID;
    x = y;
}

#if defined(REPORT_BLOSSOMS)
    printf("%d\n", NODEID(b,net));
    printf("path2 = ");
#endif

    x = find(w, map);
    while (x != b)
    {

#if defined(REPORT_BLOSSOMS)
        printf("%d ", NODEID(x,net));
#endif

        insertLast(nodes, map->v[x]);
        Union_Sets(x, b, map);
        Make_Replicate(b, map);

        aSet = NodeLinks(x,map->v[x]->mate,net);
        add_links_from_LLD(aSet, links);
        x = map->v[x]->mate;
        nMatch++;

#if defined(REPORT_BLOSSOMS)
        printf("%d ", NODEID(x,net));
#endif

        y = find(map->v[x]->pred, map);

        Union_Sets(x, b, map);
        mMake_Replicate(b, map);

        if (x != b)
        {
            insertLast(nodes, map->v[x]);

            aSet = NODELINKS(x,map->v[x]->pred, net);
            add_links_from_LLD(aSet, links);
        }
        Enqueue(Q, net->nodes->v[x]->nodeID);
        map->v[x]->bridge = ReverseLink(net, e)->indexID;
        x = y;
    } /* while (x != b) */
    insertLast(nodes, map->v[b]);

#if defined(REPORT_BLOSSOMS)
    printf("%d \n\n", NODEID(b,net));
#endif

    if (map->v[b]->mate != INVALID && \
        nMatch/2==(sizeLLD(nodes)-1)/2)
    {
        nBlossom++;
        pseudo = PseudoMapNode(nMatch, nBlossom, nodes,\
            links, b, map);
        Enqueue(Q, pseudo->nodeID);
    }
}

```

```

        ShrinkNet(nBlossom, pseudo, net);
        UpdateShrinkDual(net->fstar->anode[pseudo->nodeID], \
            map, net);
        break;
    }
}
else /* augment path */
{
    LLD P0 = createLLD();
    LLD P1 = createLLD();
    DualChange(map);
    Find_Path(P0, map, v, hv, net);
    insertFirst(P0, (int *)w);

    Find_Path(P1, map, w, hw, net);
    setCurrentToFirst(P1);
    deleteCurrent(P1);

    while(! isEmptyLLD(P0))
    {
        setCurrentToLast(P0);
        a = (int)deleteCurrent(P0);
        b = (int)deleteCurrent(P0);
        map->v[a]->mate = b;
        map->v[b]->mate = a;
    }

    while(! isEmptyLLD(P1))
    {
        setCurrentToLast(P1);
        a = (int)deleteCurrent(P1);
        b = (int)deleteCurrent(P1);
        map->v[a]->mate = b;
        map->v[b]->mate = a;
    }

    /* stop search (while Q not empty) */
    InitQueue(Q);

    /* continue main loop */
    done = false;

    break;
}
}
}
}

result = createLLD();

/* Unshrink the blossom node */
forall_nodes(v, net)
{
    if(map->v[v]->passFlag == PSEUDO)
        UnShrinkNet(v, map, net);
}

/* The result will not show repeated links */
forall_edges(e, net)
{
    int v = source(e, net);
    int w = target(e, net);
    if ( v != w && map->v[v]->mate == w && e->passFlag == PASS)
    {
        insertLast(result, e);
        map->v[v]->mate = v;
        map->v[w]->mate = w;
    }
    if_end_quit(e, net);
}

for (i = 0; i < map->num; i++)
    DestroyQueue(map->v[i]->aDual.tao);
Free_Data((header_t *)map);
Free_Data((header_t *)net->nodes);
free(net->fstar->anode);
free(net->fstar->bnode);
free(net);

```

```

    return result;
}

/*****
/* Title:      Write_Matching
/*
/* Description:
/*      Write out matching result
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      outMatch      Output file
/*      result         A matching list
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
*****/
void Write_Matching(FILE * outMatch, LLD result)
{
    ptr_link_t e;
    float distance =0.0;

    setCurrentToFirst(result);
    while (result->current != NULL)
    {
        e = getCurrentData(result);
        fprintf(outMatch, "%d %d \n", e->anode, e->bnode);
        setCurrentToNext(result);
        distance +=e->distance;
    }
    fprintf(outMatch, "The total distance is %f. \n", distance);
    destroyLLD(&result);
}

```

```

/*-----*/
/*
/* TITLE:
/*      buildtour.h
/*
/*
/* DESCRIPTION:
/*      A tour header file
/*
/*
/* ORIGIN:
/*      Programmer: Xiaohong Xin
/*      Date:      20 February 1998
/*      Platform:  Windows 95
/*      Compiler:  Borland C++ 5.0
/*
/*-----*/
#ifndef _ROUTE_H_
#define _ROUTE_H_

typedef struct CMROUTE
{
    int                routeID;
    float              cost;
    int                depot;
    float              distance;
    int                carrier;
    LLD                legs;
} cm_route_t, *ptr_cm_route_t;

typedef struct CMROUTES
{
    int                max;
    int                size;
    int                num;
    cm_route_t** v;
} cm_routes_t, *ptr_cm_routes_t;

extern BOOLEANs      Depot_Exist(int, ptr_net_t);
extern LLD           Init_Legs (int, ptr_net_t);
extern BOOLEANs     Check_Demand_Upbound(ptr_net_t, float);
extern LLD           Generate_Odds(ptr_net_t net);
extern float         Check_Lowerbound(ptr_net_t net, float, int);
extern LLD           OrderLegs(LLD);
extern void          Augment_Legs (LLD);
extern LLD           Label_Clean_Legs(LLD, ptr_net_t, int, int, int, float, int);
extern void          Init_Label(LLD, ptr_net_t);
extern int           Legs_TraversedNum(LLD);
extern void          CheckOneLeg_Demand(LLD, LLD, ptr_net_t, int, float);
extern int           Leg_Demand(LLD);
extern int           Legs_Demand(LLD);
extern float         Legs_Distance(LLD);
extern BOOLEANs     Demand_Upbound(ptr_net_t, float);
extern int           OneLeg_TraversedNum(LLD);
extern int           Legs_TraversedNum(LLD);
extern LLD           Merge_Legs (LLD, int, ptr_net_t, float, int);
extern head_tail_t * Find_Merge_Node(LLD);
extern LLD           Generate_One_Optimal_Legs(LLD, ptr_net_t, float, int);
extern void          Write_Route(FILE *, ptr_cm_route_t);
extern int           AddLinkToLeg(int, int, LLD, ptr_net_t, ptr_pathnodes_t);
extern void          TestLeg(FILE *, LLD);
extern void          TestOneLeg(LLD);
extern void          Append_Head (head_tail_t * header, head_t *);
extern void          Append_Tail (head_tail_t * header, tail_t *);
extern void          Init_Head_Tail(void);
extern void          Destroy_Merge_Node(head_tail_t **);
extern void          Free_Route(ptr_net_t, ptr_cm_routes_t);
extern void          Nodes_File(FILE *, ptr_net_t);
extern void          Links_File(FILE *, ptr_net_t);
#endif

```



```

/*-----*/
/*
/* TITLE:
/*   tour.c
/*   (arc routing problem solution and lower bound calculation)
/*
/* DESCRIPTION:
/*   A library of an implementation of augment_merge and
/*   Pearn's algorithm
/*
/* FUNCTION NAMES:
/*   Functions for augment_merge algorithm
/*
/* -----
/*   1. BOOLEANS Depot_Exist(int, ptr_net_t);
/*   2. LLD Init_Legs(int, ptr_net_t);
/*   3. BOOLEANS Check_Demand_Upbound(ptr_net_t, float);
/*   4. LLD OrderLegs(LLD);
/*   5. void Augment_Legs(LLD);
/*   6. LLD Label_Clean_Legs(LLD, ptr_net_t,int,int, int, float, int);
/*   7. void Init_Label(LLD, ptr_net_t);
/*   8. int Legs_TraversedNum(LLD);
/*   9. void CheckOneLeg_Demand(LLD, LLD, ptr_net_t, int, float);
/*  10. int Leg_Demand(LLD);
/*  11. int Legs_Demand(LLD);
/*  12. float Legs_Distance(LLD);
/*  13. BOOLEANS Demand_Upbound(ptr_net_t, float);
/*  14. int OneLeg_TraversedNum(LLD);
/*  15. int Legs_TraversedNum(LLD);
/*  16. LLD Merge_Legs(LLD, int, ptr_net_t, float, int);
/*  17. head_tail_t * Find_Merge_Node(LLD);
/*  18. LLD Generate_One_Optimal_Legs(LLD, ptr_net_t, float, int);
/*  19. void Write_Route(FILE *, ptr_cm_route_t);
/*  20. int AddLinkToLeg(int, int, LLD, ptr_net_t, ptr_pathnodes_t);
/*  21. void TestLeg(FILE *, LLD);
/*  22. void TestOneLeg(LLD);
/*  23. void Append_Head(head_tail_t * header, head_t *);
/*  24. void Append_Tail(head_tail_t * header, tail_t *);
/*  25. head_tail_t * Init_Head_Tail(void);
/*  26. void Destroy_Merge_Node(head_tail_t **);
/*  27. void Free_Route(ptr_net_t, ptr_cm_routes_t);
/*
/*   Functions for lower bound calculation
/*
/* -----
/*  28. float Check_Lowerbound(ptr_net_t net, float, int);
/*  29. LLD Generate_Odds(ptr_net_t net);
/*  30. void Nodes_File(FILE *, ptr_net_t);
/*  31. void Links_File(FILE *, ptr_net_t);
/*  32. int Even_Odd(int)
/*
/* ORIGIN:
/*   Programmer: Xiaohong Xin
/*   Date: 20 February 1998
/*   Platform: Windows 95
/*   Compiler: Borland C++ 5.0
/*
/*-----*/

```

```

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include "network.h"
#include "netlib.h"
#include "netheap.h"
#include "path.h"
#include "lld.h"
#include "buildtour.h"
#include "queue.h"
#include "matching.h"

```

```

extern int MaxNumLegs = 10000;
extern int MaxNumArcs = 10000;
extern int NumOfCarriers = 4;
extern int StartTime = 800;
extern int EndTime = 1700;
extern float Dis2Time = 45.0/100000.0;
extern int OverlayTime;
extern float FixDiscountRate;
extern void Init_Label(LLD, ptr_net_t);
extern LLD Reorder_Legs(LLD, int, int);
extern void BreakOneLeg(LLD, LLD, ptr_link_t, ptr_net_t, int);

```

```

/*****
/* Title:      Depot_Exist
/*
/* Description:
/*      Test for the existence of the depot.
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      depot        A given node ID as a depot ID
/*      net          A network with a forward star data structure
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      BOOLEANs     A boolean value
*****/
BOOLEANS Depot_Exist(int depot, ptr_net_t net)
{
    int i, j;

    for (i=0; i<net->nodes->num; i++)
    {
        j = net->nodes->v[i]->nodeID;
        if (j == depot)
            return true;
    }
    return false;
}

/*****
/* Title:      Generate_One_Optimal_Leg
/*
/* Description:
/*      Generate arc routing problem solution.
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      legs          A template linked list
/*      net          A network with a forward star data structure
/*      VehicleCapacity The vehicle capacity
/*      LastTime      An iteration number
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      tmpLegs       An output route represented as a linked list
*****/
LLD Generate_One_Optimal_Leg(LLD legs, ptr_net_t net, float VehicleCapacity, int LastTime)
{
    int j;
    LLD cLegs, newLegs, tmpLegs;

    tmpLegs = createLLD();
    tmpLegs->distance = INFINITY;

    for (j = 0; j < sizeLLD(legs); j++)
    {
        cLegs = copyLegs(legs);
        setCurrentToFirst(cLegs);
        newLegs = Merge_Legs(cLegs, j, net, VehicleCapacity, LastTime);
        destroyLegs (&cLegs);
        tmpLegs = Find_Min_Legs(tmpLegs, newLegs);
    }

    return (tmpLegs);
}

/*****
/* Title:      Init_Legs
/*
/* Description:
/*      Initialize routes.
/*
/* Inputs:
/*      Name          Description

```

```

/* ----- */
/* depot      The depot id */
/* net        A network with a forward star data structure */
/* ----- */
/* Outputs:   */
/* Name       Description */
/* ----- */
/* legs       An output route represented as a linked list */
/*****/
LLD Init_Legs(int depot, ptr_net_t net)
{
    int i;
    float distance;
    float RouteDistance = 0.0;
    static int routeCount = 0;
    LLD legs = createLLD();
    LLD leg;
    ptr_link_t aLink;
    ptr_pathnodes_t pathnodes;

    setCurrentToFirst(legs);
    for (i = 0; i < net->links->num; i++)
    {
        if (net->links->v[i]->trashamount > 0.0 && net->links->v[i]->uniqueFlag == INVALID)
        {
            net->links->v[i]->uniqueFlag = UNIQUE;
            ReverseLink(net, net->links->v[i]->uniqueFlag = UNIQUE;

            leg = createLLD();
            distance = 0;

            pathnodes = ClearOutTree(net->links->v[i]->bnode, net);
            if (FindPath(net->links->v[i]->bnode, depot, net, pathnodes)
                {
                    setCurrentToFirst(leg);
                    AddLinkToLeg(net->links->v[i]->bnode, depot, leg, net, pathnodes);
                    distance += pathnodes->v[net->fstar->anode[depot]]->distance;
                }

            Free_Data((header_t *)pathnodes);
            aLink = CopyLink(net->links->v[i]);
            insertFirst(leg, aLink);
            setCurrentToFirst(leg);
            distance += net->links->v[i]->distance;

            pathnodes = ClearOutTree(depot, net);
            if (FindPath(depot, net->links->v[i]->anode, net, pathnodes)
                {
                    AddLinkToLeg(depot, net->links->v[i]->anode, leg, net, pathnodes);
                    distance += pathnodes->v[net->fstar->anode[net->links->v[i]->anode]]->distance;
                }

            Free_Data((header_t *)pathnodes);
            leg->distance = distance;
            leg->time = distance/Dis2Time;
            insertLast(legs, leg);
        }
    }
    return (legs);
}

/*****/
/* Title:      OrderLegs */
/* ----- */
/* Description: Order routes in a decending order */
/* ----- */
/* Inputs:     */
/* Name       Description */
/* ----- */
/* mixedLegs  A linked list */
/* ----- */
/* Outputs:   */
/* Name       Description */
/* ----- */
/* legs       An output route represented as a linked list */
/*****/
LLD OrderLegs(LLD mixedLegs)
{
    LLD legs, damn, tmpLeg, leg;

```

```

legs = createLLD();

    setCurrentToFirst(legs);
setCurrentToFirst(mixedLegs);

while (mixedLegs->current != NULL)
{
    leg = getCurrentData(mixedLegs);
    damn = getCurrentData(legs);
    if (damn==NULL)
    {
        insertFirst(legs, leg);
        setCurrentToLast(legs);
    }
    else
    {
        if (leg->distance <= damn->distance)
            insertAfterCurrent(legs, leg);
        else
        {
            setCurrentToFirst(legs);
            damn = getCurrentData(legs);
            if (leg->distance >= damn->distance)
                insertFirst(legs, leg);
            else
            {
                do
                {
                    setCurrentToNext(legs);
                    tmpLeg = getCurrentData(legs);
                }while(leg->distance <= tmpLeg->distance);
                setCurrentToPrev(legs);
                insertAfterCurrent(legs, leg);
            }
        }
        setCurrentToLast(legs);
    }
    setCurrentToNext(mixedLegs);
}
return (legs);
}

/*****
/* Title:      Augment_Legs
/*
/* Description:
/*      Augment routes.
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      mixedLegs    A linked list
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
*****/
void Augment_Legs (LLD legs)
{
    LLD tmpData1, tmpData2;
    ptr_link_t tmpLink1, tmpLink2;
    setCurrentToFirst(legs);

    while (legs->current != NULL)
    {
        tmpData1 = getCurrentData(legs);
        setCurrentToFirst(tmpData1);
        tmpLink1 = getCurrentData(tmpData1);
        setCurrentToNext(legs);
        while (legs->current != NULL)
        {
            setCurrentToFirst(tmpData1);
            tmpData2 = getCurrentData(legs);
            setCurrentToFirst(tmpData2);
            tmpLink2 = getCurrentData(tmpData2);
            while (tmpLink1->arc_id==tmpLink2->arc_id && tmpData1->current!=NULL && tmpData2-
>current!=NULL)
            {
                setCurrentToNext(tmpData1);
                if (tmpData1->current != NULL)

```

```

        tmpLink1 = getCurrentData(tmpData1);
        setCurrentToNext(tmpData2);
        if (tmpData2->current != NULL)
            tmpLink2 = getCurrentData(tmpData2);
    }

    if (tmpData2->current == NULL&&(tmpData1->current != NULL||tmpData1->current==NULL))
    {
        setCurrentToPosition(legs, tmpData2);
        deleteCurrentList(legs);
        setCurrentToNext(legs);
        continue;
    }

        if (tmpData1->current == NULL&&tmpData2->current != NULL)
        {
            setCurrentToPosition(legs, tmpData1);
            deleteCurrentList(legs);
            setCurrentToNext(legs);
            break;
        }
        setCurrentToNext(legs);
    }
    setCurrentToPosition(legs, tmpData1);
    setCurrentToNext(legs);
}
return;
}

```

```

/*****/
/* Title:      Label_Clean_Legs                               */
/*                                                    */
/* Description:                                         */
/*      Order routes in a decending order                */
/*                                                    */
/* Inputs:                                             */
/*      Name          Description                        */
/*      -----          -----                        */
/*      tmpLegs       A linked list                     */
/*      net           A network with a forward star data structure */
/*      oneID         Cardinal number                  */
/*      twoID         Cardinal number                  */
/*      depot         The depot id                     */
/*      VehicleCapacity The vehicle capacity           */
/*      LastTime      An iteration number              */
/*                                                    */
/* Outputs:                                            */
/*      Name          Description                        */
/*      -----          -----                        */
/*      legs          An output route represented as a linked list */
/*****/
LLD Label_Clean_Legs (LLD tmplegs, ptr_net_t net,int oneID,int twoID, int depot,float
VehicleCapacity,int LastTime)
{
    int          *way, i, j;
    float        distance_sum = 0.0;
    LLD          damnLLD, legs;
    ptr_link_t   damnLink;

    Init_Label(tmplegs, net);
    legs = tmplegs;
    if ((way = (int *) malloc (MaxNumLegs * sizeof (int))) == NULL)
    {
        printf ("Out of Memory in Allocating Memory to Array Way!\n");
        exit(1);
    }
    i = 0;
    setCurrentToFirst(legs);
    while (legs->current != NULL)
    {
        damnLLD = getCurrentData(legs);
        distance_sum += damnLLD->distance;
        setCurrentToFirst(damnLLD);
        while (damnLLD->current != NULL)
        {
            damnLink = getCurrentData(damnLLD);
            if(net->links->v[damnLink->indexID]->trashamount > 0.0)
            {

```

```

damnLink->legFlag==INVALID)
    if (net->links->v[damnLink->indexID]->legFlag != TRAVERSED &&
        {
            for (j=0; j<i; j++)
            {
                if (damnLink->arc_id != way[j] )
                    continue;
            }
            else
                break;
        }
    if (j == i)
    {
        damnLink->legFlag = TRAVERSED;
        net->links->v[damnLink->indexID]->legFlag = TRAVERSED;
        way[i] = damnLink->arc_id;
        i += 1;
    }
}
setCurrentToNext(legs->current->data);

/*delete the nontraversed legs */
damnLLD->numTraversed = OneLeg_TraversedNum(damnLLD);
if (damnLLD->numTraversed == 0)
{
    distance_sum -= damnLLD->distance;
    deleteCurrentList(legs);
}
setCurrentToNext(legs);

setCurrentToFirst(legs);
while (legs->current != NULL)
{
    CheckOneLeg_Demand(legs, getCurrentData(legs), net, depot, VehicleCapacity);
    setCurrentToNext(legs);
}

setCurrentToFirst(legs);
while (legs->current != NULL)
{
    damnLLD = getCurrentData(legs);
    if (damnLLD->time > LastTime)
    {
        printf("Error: There are long arcs beyond the time limit! \n You must adjust
the daily working time.\n");
        exit(1);
    }
    setCurrentToNext(legs);
}

legs->demand = Legs_Demand(legs);
legs->distance = distance_sum;
legs->time = distance_sum/Dis2Time;
legs->numTraversed = Legs_TraversedNum(legs);
free(way);
return (legs);
}

/*****
/* Title:      Init_Label
/*
/* Description:
/*      label routes
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      legs      A linked list
/*      net        A network with a forward star data structure
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      void      No return value
*****/
void Init_Label(LLD legs, ptr_net_t net)
{

```

```

LLD damnLLD;
ptr_link_t damnLink;

setCurrentToFirst(legs);
while (legs->current != NULL)
{
    damnLLD = getCurrentData(legs);
    setCurrentToFirst(damnLLD);
    while (damnLLD->current != NULL)
    {
        damnLink = getCurrentData(damnLLD);
        damnLink->legFlag = INVALID;
        net->links->v[damnLink->indexID]->legFlag = INVALID;
        setCurrentToNext(damnLLD);
    }
    setCurrentToNext(legs);
}
return;
}

/*****
/* Title:      Reorder_Legs
/*
/* Description:
/*      reorder routes according to two cardinal numbers.
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      legs          A linked list
/*      oneID         Cardinal number
/*      twoID         Cardinal number
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      newLegs       An output route represented as a linked list
*****/
LLD Reorder_Legs(LLD legs, int oneID, int twoID)
{
    LLD newLegs, oneData, twoData, elseData;
    newLegs = createLLD();
    oneData = getOrderData(legs, oneID);
    insertLast(newLegs, oneData);
    twoData = getOrderData(legs, twoID);
    insertLast(newLegs, twoData);

    setCurrentToFirst(legs);
    while (legs->current != NULL)
    {
        elseData = getCurrentData(legs);
        if (elseData != oneData && elseData != twoData)
            insertLast(newLegs, elseData);
        setCurrentToNext(legs);
    }

    return (newLegs);
}

/*****
/* Title:      Demand_Upbound
/*
/* Description:
/*      Check for link demand
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      net           A network with a forward star data structure
/*      VehicleCapacity The vehicle capacity
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      BOOLEANs     A boolean output
*****/
BOOLEANs Demand_Upbound(ptr_net_t net, float VehicleCapacity)
{
    int i;
    int max = INFINITY*(-1);

```

```

        for (i = 0; i <net->links->num; i++)
            if (net->links->v[i]->trashamount > max)
                max = net->links->v[i]->trashamount;

    if (max > VehicleCapacity)
        return (false);
    else
        return (true);
}

/*****
/* Title:      Nodes_File
/*
/* Description:
/*      Generate a matching node file
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      Ndfile        An output node file
/*      net           A network with a forward star data structure
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
*****/
void Nodes_File(FILE *NdFile, ptr_net_t net)
{
    int i;

    fprintf(NdFile, "#Knoxroad\n");

    for (i=0; i<net->nodes->num; i++)
    {
        if (Even_Odd(net->nodes->v[i]->undirectedLinks)== 1)
            fprintf(NdFile, "%d\n", net->nodes->v[i]->nodeID);
    }
    fclose(NdFile);
}

/*****
/* Title:      Links_File
/*
/* Description:
/*      Generate a matching link file
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      Lkfile        An output link file
/*      net           A network with a forward star data structure
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
*****/
void Links_File(FILE * LkFile, ptr_net_t net)
{
    int *Odd, i, j, k, arcID=0;
    ptr_pathnodes_t pathnodes;
    ptr_link_t aLink, bLink;

    if ((Odd = (int *) malloc ((net->nodes->num)* sizeof (int))) == NULL)
    {
        printf ("Out of Memory in Allocating Memory to Array Odd!\n");
        exit(1);
    }

    k = 0;
    for (i=0; i<net->nodes->num; i++)
    {
        if (Even_Odd(net->nodes->v[i]->undirectedLinks)== 1)
            Odd[k++] = net->nodes->v[i]->nodeID;
    }

    fprintf(LkFile, "#Fnode_, #Tnode_, #Length, #Knoxroad_, #Fromlong, #Fromlat, #Tolong, #Tolat,
#Two_way, #Demand\n");

```



```

for (i=0; i<k; i++)
  for (j=0; j<k; j++)
  {
    if (j > i)
    {
      pathnodes = ClearOutTree(Odd[i], net);
      if (FindPath(Odd[i], Odd[j], net, pathnodes))
        aLink=net->links->v[net->fstar->bnode[net->fstar->anode[Odd[i]]]];
        bLink=net->links->v[net->fstar->bnode[net->fstar->anode[Odd[j]]]];
        fprintf(LkFile, "%d,%d,%f,%d,%ld,%ld,%ld,%ld,%hd,%f\n",
            Odd[i], Odd[j], pathnodes->v[net->fstar->anode[Odd[j]]]->distance,
            arcID++, aLink->org_node.lon, aLink->org_node.lat, bLink->dst_node.lon,
            bLink->dst_node.lat, aLink->two way, aLink->trashamount);
            Free_Data((header_t *)pathnodes);
    }
  }
free(Odd);
fclose(LkFile);
}

/*****
/* Title:      Even_Odd
/*
/* Description:
/*      judge a number as even or odd?
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      number    An integer number
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      k         An integer as a boolean number
*****/
int Even_Odd (int number)
{
    int k;
    k = number%2;
    while (k != 0 && k != 1)
    {
        k = k%2;
    }
    return k;
}

/*****
/* Title:      Generate_Odds
/*
/* Description:
/*      generate odd-degree node set and then do matching for the nodes
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      net       A network with a forward star data structure
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      legs     An output route represented as a linked list
*****/
LLD Generate_Odds(ptr_net_t net)
{
    FILE *NdFile, *LkFile, *netfile, *matchfile;
    ptr_net_t oddnet;
    LLD matchodds;

    /*generate odd-degree node net */
    NdFile = Open_File("mnode.txt", "w");
    Nodes_File(NdFile, net); /*read odd-degree nodes in */
    LkFile = Open_File("mlink.txt", "w");
    Links_File(LkFile, net); /* read the links for the odd nodes */
    NetBuild("mnode.txt", "mlink.txt", "oddnet.txt");
    netfile = Open_File("oddnet.txt", "r");
    oddnet = Init_Network();
    ReadNet(netfile, oddnet);
}

```

```

    /*match odd-degree nodes; */
    matchodds = max_card_matching(oddnet, 2);

    return matchodds;
}

/*****
/* Title:      Check_Lowerbound */
/*
/* Description:
/*      Lower bound calculation.
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      net           A network with a forward star data structure
/*      VehicleCapacity The vehicle capacity
/*      depot        The depot id
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      lbD          lower bound
*****/
float Check_Lowerbound (ptr_net_t net, float VehicleCapacity, int depot)
{
    int i, j, k, I, dest, nL=0;
    float totalDistance = 0.0, totalDemand = 0.0, spl = 0.0, distance = 0.0, lbD = 0.0;
    ptr_pathnodes_t pathnodes;
    ptr_pair_t aPair;
    ptr_link_t e;
    ptr_minheap_t aheap = createHeap(net->nodes->num);
    LLD matchnodes = Generate_Odds(net);

    /*compute the double total distance */
    for (i=0; i<net->links->num; i++)
    {
        totalDemand += net->links->v[i]->trashamount;
        if (net->links->v[i]->trashamount > 0.0)
            totalDistance += net->links->v[i]->distance;
    }

    /*compute I */
    I = totalDemand/VehicleCapacity-net->nodes->v[net->fstar->anode[depot]]->outLinks;

    for (j=0; j<net->nodes->num; j++)
    {
        dest = net->nodes->v[j]->nodeID;
        if (dest != depot)
        {
            pathnodes = ClearOutTree(depot, net);
            if(FindPath(depot, dest, net, pathnodes))
                addVertexToHeap(aheap, j ,pathnodes->v[j]->distance);
            Free_Data((header_t *)pathnodes);
        }
    }

    /*deal with odd nodes and their paths relating with spl */
    setCurrentToFirst(matchnodes);
    while (matchnodes->current != NULL)
    {
        e = getCurrentData(matchnodes);
        pathnodes = ClearOutTree(e->anode, net);
        if(FindPath(e->anode, e->bnode, net, pathnodes))
            if ((pathnodes->v[net->fstar->anode[depot]]->parent==e->anode)||
                (pathnodes->v[net->fstar->anode[depot]]->parent==e->bnode))
                nL -= 2;
        setCurrentToNext(matchnodes);
        distance +=e->distance;
    }
    destroyLLD(&matchnodes);

    /*compute the total amount of spl */
    while (nL <= I)
    {
        aPair = heapDeleteMin(aheap);
        spl += (aPair->priority)*(net->nodes->v[aPair->vertex]->undirectedLinks);
        nL += net->nodes->v[aPair->vertex]->undirectedLinks;
    }
}

```

```

        destroyHeap(aheap);

    /*finally, the ideal lower bound comes as follows */
    lbD = totalDistance/2 + distance + spl;

    return lbD;
}

/*****
/* Title:      CheckOneLeg_Demand
/*
/* Description:
/*      Check demand capacity for one route
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      legs          all routes as a linked list
/*      leg           one route as a linked list
/*      net           A network with a forward star data structure
/*      depot         The depot id
/*      VehicleCapacity The vehicle capacity
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
*****/
void CheckOneLeg_Demand(LLD legs, LLD leg, ptr_net_t net, int depot, float VehicleCapacity)
/*assuming no violation of time constraints for one leg */
{
    ptr_link_t aLink;
    int tAmount = 0;

    setCurrentToFirst(leg);
    while (leg->current != NULL)
    {
        aLink = getCurrentData(leg);
        if (aLink->legFlag == TRAVERSED)
            tAmount += aLink->trashamount;
        if (tAmount > VehicleCapacity)
        {
            setCurrentToPrev(leg);
            BreakOneLeg(legs, leg, getCurrentData(leg), net, depot);
            return;
        }
        setCurrentToNext(leg);
    }

    if (leg->current == NULL)
        leg->demand = tAmount;
    return;
}

/*****
/* Title:      OneLeg_TraversedNum
/*
/* Description:
/*      Obtain the number of links traversed
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      theLeg        one route as a linked list
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      tNum          the number of traversed links on one route
*****/
int OneLeg_TraversedNum(LLD theLeg)
{
    int tNum = 0;
    ptr_link_t theLink;

    setCurrentToFirst(theLeg);
    while (theLeg->current != NULL)
    {
        theLink = getCurrentData(theLeg);
        if (theLink->legFlag == TRAVERSED)

```

```

        tNum += 1;
        setCurrentToNext(theLeg);
    }
    return (tNum);
}

/*****
/* Title:      Legs_TraversedNum
/*
/* Description:
/*      Obtain the number of links traversed for all routes
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      legs          all routes as a linked list
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      tNum          the number of traversed links on all routes
*****/
int Legs_TraversedNum(LLD legs)
{
    int tNum = 0;
    LLD theLeg;

    setCurrentToFirst(legs);
    while (legs->current != NULL)
    {
        theLeg = getCurrentData(legs);
        tNum += theLeg->numTraversed;
        setCurrentToNext(legs);
    }
    return (tNum);
}

/*****
/* Title:      BreakOneLeg
/*
/* Description:
/*      break one leg into two once routes violate capacity limit
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      legs          all routes as a linked list
/*      leg           one route as a linked list
/*      theLink       the working link
/*      net           A network with a forward star data structure
/*      depot         The depot id
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
*****/
void BreakOneLeg(LLD legs, LLD theLeg, ptr_link_t theLink, ptr_net_t net, int depot)
{
    ptr_link_t aLink, bLink, nextLink;
    LLD amergeLeg, bmergeLeg, aLeg, bLeg;
    ptr_pathnodes_t pathnodes;
    int tAmount = 0;

    setCurrentToPosition(theLeg, theLink);
    if (theLeg->current->next != NULL)
        nextLink = theLeg->current->next->data;

    setCurrentToPosition(theLeg, theLink);
    aLeg = createLLD();
    pathnodes = ClearOutTree(theLink->bnode, net);
    if (FindPath(theLink->bnode, depot, net, pathnodes))
    {
        setCurrentToFirst(aLeg);
        AddLinkToLeg(theLink->bnode, depot, aLeg, net, pathnodes);
        aLeg->distance = pathnodes->v[net->fstar->anode[depot]]->distance;
    }

    setCurrentToFirst(aLeg);
    aLink = getCurrentData(aLeg);

```

```

    amergeLeg = merge2LLD(theLeg, aLeg, theLink, aLink);
    bLeg = createLLD();
    pathnodes = ClearOutTree(depot, net);
    if (FindPath(depot, theLink->bnode, net, pathnodes))
    {
        setCurrentToFirst(bLeg);
        AddLinkToLeg(depot, theLink->bnode, bLeg, net, pathnodes);
        bLeg->distance = pathnodes->v[net->fstar->anode[theLink->bnode]]->distance;
    }

    setCurrentToLast(bLeg);
    bLink = getCurrentData(bLeg);

    bmergeLeg = merge2LLD(bLeg, theLeg, bLink, nextLink);

    insertFirst(legs, bmergeLeg);
    insertFirst(legs, amergeLeg);
    setCurrentToPosition(legs, theLeg);
    deleteCurrentList(legs);
    setCurrentToFirst(legs);
}

/*****
/* Title:      Leg_Demand
/*
/* Description:
/*      calculate the demand on one route
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      aLeg      one route as a linked list
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      legDemand the amount of demand
*****/
int Leg_Demand(LLD aLeg)
{
    int legDemand = 0;
    ptr_link_t aLink;

    setCurrentToFirst(aLeg);
    while (aLeg->current != NULL)
    {
        aLink = getCurrentData(aLeg);
        if (aLink->legFlag == TRAVERSED)
            legDemand += aLink->trashamount;
        setCurrentToNext(aLeg);
    }
    aLeg->demand = legDemand;
    return (aLeg->demand);
}

/*****
/* Title:      Legs_Distance
/*
/* Description:
/*      calculate the distance on all routes
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      legs      all routes as a linked list
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      legDistance distance
*****/
float Legs_Distance(LLD legs)
{
    float legsDistance = 0.0;
    LLD aLeg;

    setCurrentToFirst(legs);
    while (legs->current != NULL)

```

```

    {
        aLeg = getCurrentData(legs);
        legsDistance += aLeg->distance;
        setCurrentToNext(legs);
    }
    legs->distance = legsDistance;
    return (legs->distance);
}

```

```

/*****
/* Title:      Legs_Demand
/*
/* Description:
/*      calculate the demand on all routes
/*
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      legs      all routes as a linked list
/*
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      legsDemand  the amount of demand
*****/

```

```

int Legs_Demand(LLD legs)
{
    int legsDemand = 0;
    LLD aLeg;

    setCurrentToFirst(legs);
    while (legs->current != NULL)
    {
        aLeg = getCurrentData(legs);
        legsDemand += aLeg->demand;
        setCurrentToNext(legs);
    }
    legs->demand = legsDemand;
    return (legs->demand);
}

```

```

/*****
/* Title:      Merge_Legs
/*
/* Description:
/*      merge routes
/*
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      origlegs  all routes as a linked list
/*      legOrder  cardinal number
/*      net       A network with a forward star data structure
/*      depot     The depot id
/*      VehicleCapacity  The vehicle capacity
/*      LastTime  An iteration number
/*
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      minLLDs   An output routes
*****/

```

```

LLD Merge_Legs(LLD origlegs, int legOrder, ptr_net_t net, float VehicleCapacity, int LastTime)
{
    int i, j, k, nTail, f;
    int cnt = 0;

    ptr_head_tail_t m1=NULL, m2=NULL;
    LLD legData1 = NULL, legData = NULL, newLegs, newLLD, minLLD, minLLDs, origData, legs,
tmpData;
    ptr_link_t damnTailLink, damnHeadLink, m1Link, m2Link;

    minLLDs = createLLD();
    minLLDs->distance = INFINITY;
    origData = getOrderData(origlegs, legOrder);
    damnTailLink = origData->tail->data;
    if (damnTailLink->legFlag != TRAVERSED)
    {

```

```

        m1 = Find_Merge_Node(origData);
nTail = m1->NumTails;

        for (i = 0; i < nTail; i++)
        {
            legs = copyLegs(origlegs);
            tmpData = getOrderData(legs, legOrder);
            Destroy_Merge_Node(&m1);
            m1 = Find_Merge_Node(tmpData);
            setCurrentToPosition(legs, tmpData);
            setCurrentToNext(legs);
            while (legs->current != NULL)
            {
                minLLD = createLLD();
                minLLD->distance = INFINITY;

                for (k=0; k<2; k++)
                {
                    legData = getCurrentData(legs);
                    if (k==1)
                        legData = getReverseList(net, legData);
                    Destroy_Merge_Node(&m2);
                    m2 = Find_Merge_Node(legData);
                    for (j=0; j<m2->NumHeads; j++)
                    {
                        if (m1->NumTails!=0&&m2->NumHeads!=0)
                        {
                            for (f=0; f<m1->NumTails; f++)
                            {
                                m1Link = m1->mergeTailNodes[f]-
                                m2Link = m2->mergeHeadNodes[j]-
                                if (m1->mergeTailNodes[f]-
                                {
                                    newLLD =
                                merge2LLD(tmpData, legData, m1->mergeTailNodes[f]->taillink, m2->mergeHeadNodes[j]->headlink);
                                if ((newLLD->demand >
                                VehicleCapacity) ||
                                (newLLD->time >
                                LastTime))
                                {
                                    destroyLLD
                                }
                                minLLD =
                                continue;
                            }
                        }
                    }
                    /*for (f=0; f<m1->NumTails; f++) */
                    /*if (m1->NumTails!=0&&m2->NumHeads!=0) */
                    } /*for (j=0; j<m2->NumHeads; j++) */
                    if (k==1)
                        destroyLLD (&legData);
                } /*for (k=0; k<2; k++) */

                /*finally a better leg comes */
                if (minLLD->distance < INFINITY)
                {
                    deleteCurrentList(legs); /* sizeof(legs) changed */
                    setCurrentToPosition(legs, tmpData);
                    deleteCurrentList(legs);
                    insertFirst(legs, minLLD);
                    setCurrentToFirst(legs);
                    tmpData = getCurrentData(legs);
                    Destroy_Merge_Node(&m1);
                    m1 = Find_Merge_Node(tmpData);
                    setCurrentToFirst(legs);
                    if (m1->NumTails == 0)
                    {
                        setCurrentToPosition(legs, tmpData);
                        tmpData = getReverseList(net, tmpData);
                        deleteCurrentList(legs);
                        insertFirst(legs, tmpData);
                        Destroy_Merge_Node(&m1);
                        m1 = Find_Merge_Node(tmpData);
                        if (m1->NumTails == 0)
                            break;
                    }
                }
            }
        }
        Find_Min_Leg(minLLD, newLLD);
    }
}

```

```

else
    setCurrentToFirst(legs);
    /* if (ml->NumTails == 0) */
    /* if (minLLD->distance != INFINITY) */
else
    destroyLLD (&minLLD);

    setCurrentToNext(legs);
} /*while (legs->current != NULL) */

legs->distance = Legs_Distance(legs);
legs->time = (legs->distance)/Dis2Time;
legs->demand = Legs_Demand(legs);
legs->numTraversed = Legs_TraversedNum(legs);
minLLDs = Find_Min_Legs(minLLDs, legs);
Destroy_Merge_Node (&m2);

) /*for (i=0; i<ml->NumTails; i++) */
} /*if (damnTailLink->legFlag != TRAVERSED) */
Destroy_Merge_Node (&m1);
return (minLLDs);
}

/*****
/* Title:      Find_Merge_Node
/*
/* Description:
/*      build merge lists
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      aLeg          one route as a linked list
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      newHead_Tail  A head_tail list
*****/
head_tail_t * Find_Merge_Node(LLD aLeg)
{
    float saving_Head_distance = 0.0, saving_Tail_distance = 0.0;
    ptr_link_t damnTailLink, damnHeadLink, damnCurLink;
    head_t * newHead;
    tail_t * newTail;
    head_tail_t * newHead_Tail;

    newHead_Tail = Init_Head_Tail();
    setCurrentToFirst(aLeg);
    if (aLeg->current != NULL)
    {
        damnTailLink = aLeg->tail->data;
        damnHeadLink = aLeg->tail->next->data;
    }

    if (damnTailLink->legFlag == INVALID)
    {
        setCurrentToLast(aLeg);
        if (aLeg->current != NULL)
            damnCurLink = getCurrentData(aLeg);
        saving_Tail_distance = damnCurLink->distance;

        setCurrentToPrev(aLeg);
        if (aLeg->current != NULL)
            damnCurLink = getCurrentData(aLeg);
        while (damnCurLink->legFlag!=TRAVERSED && aLeg->current != NULL)
        {
            newTail = (tail_t *)New_Element(sizeof(tail_t));
            newTail->tailnode = damnCurLink->bnode;
            newTail->taillink = damnCurLink;
            newTail->saving_Tail_distance = saving_Tail_distance;

            Append_Tail(newHead_Tail, newTail);
            saving_Tail_distance += damnCurLink->distance;
            setCurrentToPrev(aLeg);
            damnCurLink = getCurrentData(aLeg);
        }
        newTail = (tail_t *)New_Element(sizeof(tail_t));
        newTail->tailnode = damnCurLink->bnode;
        newTail->taillink = damnCurLink;
    }
}

```



```

newTail->saving_Tail_distance = saving_Tail_distance;

Append_Tail(newHead_Tail, newTail);
}

if (damnHeadLink->legFlag == INVALID)
{
    setCurrentToFirst(aLeg);
    if (aLeg->current != NULL)
    damnCurLink = getCurrentData(aLeg);
    saving_Head_distance = damnCurLink->distance;

    setCurrentToNext(aLeg);
    if (aLeg->current != NULL)
    damnCurLink = getCurrentData(aLeg);
    while(damnCurLink->legFlag == INVALID && aLeg->current != NULL)
    {
        newHead = (head_t *)New_Element(sizeof(head_t));
        newHead->headnode = damnCurLink->anode;
        newHead->headlink = damnCurLink;
        newHead->saving_Head_distance = saving_Head_distance;

        Append_Head(newHead_Tail, newHead);

        saving_Head_distance += damnCurLink->distance;
        setCurrentToNext(aLeg);
        damnCurLink = getCurrentData(aLeg);
    }

    newHead = (head_t *)New_Element(sizeof(head_t));
    newHead->headnode = damnCurLink->anode;
    newHead->headlink = damnCurLink;
    newHead->saving_Head_distance = saving_Head_distance;

    Append_Head(newHead_Tail, newHead);
}

aLeg->mergeNodes = newHead_Tail;
return (aLeg->mergeNodes);
}

/*****
/* Title:      Destry_Merge_Node
/*
/* Description:
/*      free merge lists
/*
/* Inputs:
/*      Name          Description
/*      -----
/*      m              A head_tail list
/*
/* Outputs:
/*      Name          Description
/*      -----
/*      void          No return value
*****/
void Destroy_Merge_Node(head_tail_t ** m)
{
    int i, j;
    if (*m == NULL)
        return;
    for (i=0; i < (*m)->NumHeads; i++)
        free ((*m)->mergeHeadNodes[i]);
    free ((*m)->mergeHeadNodes);
    for (j = 0; j < (*m)->NumTails; j++)
        free ((*m)->mergeTailNodes[j]);
    free ((*m)->mergeTailNodes);
    free(*m);
    *m = NULL;
}

/*****
/* Title:      Write_Route
/*
/* Description:
/*      write out arc routing solution
/*
/* Inputs:
/*      Name          Description
*****/

```

```

/* ----- */
/* outfile      An output file */
/* route        An arc routing solution */
/* ----- */
/* Outputs: */
/* Name          Description */
/* ----- */
/* void          No return value */
/*****
void Write_Route(FILE * outfile, ptr_cm_route_t route)
{
    int i, j;
    LLD damnLLD;
    ptr_link_t damnLink;
    fprintf(outfile, "%d\n", route->carrier);
    fprintf(outfile, "%d\n", route->depot);

    setCurrentToFirst(route->legs);
    for(i=0; i<sizeLLD(route->legs); i++)
    {
        fprintf(outfile, "%d\n", i);
        setCurrentToFirst(route->legs->current->data);
        for (j=0; j<sizeLLD(route->legs->current->data); j++)
        {
            damnLLD = route->legs->current->data;
            damnLink = damnLLD->current->data;
            fprintf(outfile, "%d\n", damnLink->indexID);
            setCurrentToNext(damnLLD);
        }
        setCurrentToNext(route->legs);
    }
}

/*****
/* Title:      AddLinkToLeg */
/* Description: Build one route from paths */
/* ----- */
/* Inputs: */
/* Name          Description */
/* ----- */
/* start        path start index */
/* end          path end index */
/* leg          a route as a linked list */
/* net          A network with a forward star data structure */
/* LastPath     A path structure */
/* ----- */
/* Outputs: */
/* Name          Description */
/* ----- */
/* TRUE?FALSE   An integer represented as a boolean */
/*****
int AddLinkToLeg(int start, int end, LLD leg, ptr_net_t net, ptr_pathnodes_t LastPath)
{
    int cur;
    ptr_link_t aLink;
    cur = end;
    if (end == start)
        return (TRUE);
    else
    {
        while (cur != start)
        {
            aLink = CopyLink(net->links->v[LastPath->v[net->fstar->anode[cur]]->indexID]);
            aLink->legFlag = INVALID;
            insertFirst(leg, aLink);
            cur = LastPath->v[net->fstar->anode[cur]]->parent;
        }
        setCurrentToFirst(leg);
        return (TRUE);
    }
}

/*****
/* Title:      Test_Leg */
/* Description: print out attributes on one leg */

```

```

/* Inputs:
/* Name Description
/* -----
/* .outfile An output file
/* legs All routes
/*
/* Outputs:
/* Name Description
/* -----
/* void No return value
/*****
void TestLeg(FILE * outfile, LLD legs)
{
    LLD damn;
    ptr_link_t dam;

    fprintf(outfile, "The total distance is %lf\n", legs->distance);
    fprintf(outfile, "The total time is %lf\n", legs->time);
    fprintf(outfile, "The total demand is %f\n", legs->demand);
    fprintf(outfile, "The total traversed link number is %d\n", legs->numTraversed);

    setCurrentToFirst(legs);
    while (legs->current != NULL)
    {
        damn = getCurrentData(legs);
        fprintf(outfile, "\nThe leg distance is %lf\n", damn->distance);
        fprintf(outfile, "The leg time is %lf\n", damn->time);
        fprintf(outfile, "The leg demand is %f\n", damn->demand);
        fprintf(outfile, "The leg traversed link number is %d\n", damn->numTraversed);

        setCurrentToFirst(damn);
        while (damn->current != NULL)
        {
            dam = getCurrentData(damn);
            fprintf(outfile, "%d %d %f %d %d %d %lf %lf\n", dam->anode, dam->bnode, dam->trashamount,
dam->indexID, dam->legFlag, dam->arc_id, dam->distance, dam->time);
            setCurrentToNext(damn);
        }
        setCurrentToNext(legs);
    }
}

/*****
/* Title: TestOneLeg
/*
/* Description:
/* print out attributes on one leg
/*
/* Inputs:
/* Name Description
/* -----
/* aLeg One Route
/*
/* Outputs:
/* Name Description
/* -----
/* void No return value
/*****
void TestOneLeg(LLD aLeg)
{
    ptr_link_t dam;
    int cnt = 0;

    printf ("BEGIN: TestOneLeg\n");
    printf ("aLeg->length = %d\n", aLeg->length);

    if (aLeg == NULL)
    {
        printf ("leg is empty\n");
        return;
    }
    else
    {
        printf("The leg distance is %lf\n", aLeg->distance);
        printf("The leg time is %lf\n", aLeg->time);
        printf("The leg demand is %f \n", aLeg->demand);
        printf("The leg traversed link number is %d \n", aLeg->numTraversed);

        setCurrentToFirst(aLeg);

```

```

    while (aLeg->current != NULL)
    {
        dam = getCurrentData(aLeg);
        printf("%d %d %d %d %f %d %d %lf %lf\n", cnt++, dam->anode, dam->bnode, dam->legFlag, dam-
>trashamount, dam->indexID, dam->arc_id, dam->distance, dam->time);
        setCurrentToNext(aLeg);
    }
}

printf ("END: TestOneLeg\n");
}

```

```

/*****
/* Title:      Append_Head
/*
/* Description:
/*      Build a dynamic head_tail array for head nodes
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      header    A dynamic array of head_tail nodes
/*      element    A dynamic array of head nodes
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      void      No return value
*****/

```

```

void Append_Head (head_tail_t * header, head_t *element)
{
    if (header->NumHeads >= MaxNumArcs)
    {
        printf ("Increase header size (Append_Head ())\n");
        exit (1);
    }

    header->mergeHeadNodes[header->NumHeads++] = element;
}

```

```

/*****
/* Title:      Append_Tail
/*
/* Description:
/*      Build a dynamic head_tail array for tail nodes
/*
/* Inputs:
/*      Name      Description
/*      -----
/*      header    A dynamic array of head_tail nodes
/*      element    A dynamic array of tail nodes
/*
/* Outputs:
/*      Name      Description
/*      -----
/*      void      No return value
*****/

```

```

void Append_Tail (head_tail_t * header, tail_t *element)
{
    if (header->NumHeads >= MaxNumArcs)
    {
        printf ("Increase header size (Append_Tail ())\n");
        exit (1);
    }

    header->mergeTailNodes[header->NumTails++] = element;
}

```

```

/*****
/* Title:      Init_Head_Tail
/*
/* Description:
/*      initialize a dynamic head_tail array
/*
/* Inputs:
/*      Name      Description
/*      -----
/*
/* Outputs:

```

```

/*      Name          Description          */
/*      -----          -----          */
/*      header_tail   A dynamic array of head_tail nodes          */
/*****
head_tail_t * Init_Head_Tail()
{
    head_tail_t * head_tail;
    if ((head_tail = (head_tail_t *) malloc (sizeof (head_tail_t))) == NULL)
    {
        printf ("Error in malloc header (Init_Head_Tail())\n");
        exit (1);
    }

    head_tail->NumHeads = 0;
    head_tail->NumTails = 0;
    if ((head_tail->mergeHeadNodes = (head_t **) calloc (MaxNumArcs, sizeof (head_t *))) == NULL)
    {
        printf ("Error in malloc header->mergeHeadNodes(Init_Head_Tail())\n");
        exit (1);
    }

    if ((head_tail->mergeTailNodes = (tail_t **) calloc (MaxNumArcs, sizeof (tail_t *))) == NULL)
    {
        printf ("Error in malloc header->mergeTailNodes(Init_Head_Tail())\n");
        exit (1);
    }
    return head_tail;
}

/*****
/* Title:      Find_Merge_Node          */
/*          */
/* Description:          */
/*      build merge lists          */
/*          */
/* Inputs:          */
/*      Name          Description          */
/*      -----          -----          */
/*      net          A network with a forward star data structure          */
/*      routes          An arc route solution          */
/*          */
/* Outputs:          */
/*      Name          Description          */
/*      -----          -----          */
/*      void          No return value          */
/*****
void Free_Route(ptr_net_t net,ptr_cm_routes_t routes)
{
    free(net->fstar->anode);
    free(net->fstar->bnode);
    free(net->fstar->cnode);
    free(net->links);
    free(net->nodes);
    free(net);
    free(routes);
}

```

Vita

Xiaohong Xin was born in Urumqi, a capital city of Xinjing Uygur Autonomous Region, P. R. China, on November 4, 1969. She attended No. 23 Middle School in Urumqi where she graduated in 1987. She received her Bachelor of Science Degree from Lanzhou University, Gansu, P. R. China, in 1991, with a major in Economic Geography and Urban & Regional Planning. She received her Master of Science Degree also from Lanzhou University, in 1994, with a major in Urban & Regional Planning. After graduating from Lanzhou University, she became an assistant professor in Lanzhou University to teach Regional Economic Geography and Urban Economics from 1994 until 1995. In 1995, she entered the Department of Geography at the University of Tennessee, Knoxville (UTK) in pursuit of a Master of Science Degree with an emphasis in the use of Geographic Information System for Transportation. Upon successful defense of this thesis, she will continue her Ph.D study in UTK.