To my parents, Kala Nemani and Vijay Nemani
ACKNOWLEDGMENTS

First, I want to express my deepest thanks to my advisor Dr. Ravindra K. Ahuja who helped me in each aspect of my life in the last four years. His guidance during my graduate study not only helped me as a researcher but also as a human being. His unique ways of encouragement always motivated us to work harder and develop a disciplined approach towards life. I would also like to thank my committee members for their help in improving my research papers by adding new dimensions.

A special "thank you" goes to my dear friends, Abhudaya Mishra, Arvind Kumar, Krishna Jha, Amit Agarwal, Ashwin Arulselwan, and Ashutosh Chohan who made my life easier by their suggestions in difficult moments. I would specifically like to mention the name of my colleague Suat Bog, with whom I worked on two chapters of my dissertation.

Finally, I would like to thank my family for their consistent emotional support. And last but not least, I would like to acknowledge the support of my brother, Alok Nemani, for his guidance throughout my life by suggesting me the optimal and implementable solutions for all problems in my path.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>8</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>9</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>10</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>1  INTRODUCTION</td>
<td>12</td>
</tr>
<tr>
<td>2  LOAD PLANNING PROBLEM AT AN INTERMODAL RAILROAD TERMINAL</td>
<td>18</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>18</td>
</tr>
<tr>
<td>2.2 Definitions and Background</td>
<td>21</td>
</tr>
<tr>
<td>2.3 Problem Description</td>
<td>24</td>
</tr>
<tr>
<td>2.3.1 Basic Operations</td>
<td>25</td>
</tr>
<tr>
<td>2.3.2 General Rules of Loading Units on Railcars</td>
<td>26</td>
</tr>
<tr>
<td>2.3.3 Hitch Utilization</td>
<td>27</td>
</tr>
<tr>
<td>2.3.4 Aerodynamic Efficiency</td>
<td>27</td>
</tr>
<tr>
<td>2.4 Model Formulation</td>
<td>28</td>
</tr>
<tr>
<td>2.5 Heuristics: Very Large Scale Neighborhood Search Algorithms</td>
<td>33</td>
</tr>
<tr>
<td>2.5.1 Construction Heuristics</td>
<td>35</td>
</tr>
<tr>
<td>2.5.2 The Neighborhood Search Structure</td>
<td>35</td>
</tr>
<tr>
<td>2.5.3 Improvement Graph</td>
<td>36</td>
</tr>
<tr>
<td>2.5.4 Cyclic Exchanges</td>
<td>37</td>
</tr>
<tr>
<td>2.5.5 Path Exchanges</td>
<td>38</td>
</tr>
<tr>
<td>2.5.6 Identifying Path Exchanges</td>
<td>38</td>
</tr>
<tr>
<td>2.6 Very Large Scale Neighborhood With Tabu Search</td>
<td>39</td>
</tr>
<tr>
<td>2.7 Hybrid Approach</td>
<td>41</td>
</tr>
<tr>
<td>2.8 Computational Results</td>
<td>41</td>
</tr>
<tr>
<td>2.8.1 Comparison of Exact Approach and Heuristics</td>
<td>42</td>
</tr>
<tr>
<td>2.8.2 Results at Different Stages of Hybrid Approach</td>
<td>44</td>
</tr>
<tr>
<td>2.9 Flexibility of the Model</td>
<td>45</td>
</tr>
<tr>
<td>2.9.1 Local Carry Cost</td>
<td>45</td>
</tr>
<tr>
<td>2.9.2 Handling Time at a Terminal</td>
<td>46</td>
</tr>
<tr>
<td>2.9.3 Daily or Weekly Load Plan</td>
<td>46</td>
</tr>
<tr>
<td>2.10 Conclusions</td>
<td>47</td>
</tr>
<tr>
<td>3  SUBSET DISJOINT MINIMUM COST CYCLE DETECTION</td>
<td>48</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>48</td>
</tr>
<tr>
<td>3.2 Network Reduction</td>
<td>52</td>
</tr>
<tr>
<td>3.3 Exact Algorithms for Subset Disjoint Minimum Cost Cycles</td>
<td>53</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Labeling Scheme</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Dynamic Labeling Algorithm (DLA) for SDMCC Problems</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Algorithm Acceleration for DLA</td>
</tr>
<tr>
<td>3.3.4</td>
<td>DLA with Acceleration for SDNCC Problems</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Fixed Labeling Algorithm (FLA) for SDMCC Problems</td>
</tr>
<tr>
<td>3.3.6</td>
<td>All-Pairs like Pull Algorithms (APPull)</td>
</tr>
<tr>
<td>3.3.7</td>
<td>All-Pairs like Push Algorithms (APPush)</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Strategy to Avoid Inefficient Label Extensions</td>
</tr>
<tr>
<td>3.4</td>
<td>Heuristic Algorithms for Subset Disjoint Minimum Cost Cycles</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Limited Unprocessed Labels Heuristics</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Limited Cycle Length Heuristics</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Limited Cycle Number Heuristics</td>
</tr>
<tr>
<td>3.5</td>
<td>Computational Analysis</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusions</td>
</tr>
<tr>
<td>4</td>
<td>COLUMN GENERATION APPROACH FOR LOCATION ROUTING PROBLEMS</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.2</td>
<td>Problem Description</td>
</tr>
<tr>
<td>4.3</td>
<td>Mathematical Model</td>
</tr>
<tr>
<td>4.4</td>
<td>Column Generation Algorithm</td>
</tr>
<tr>
<td>4.4.1</td>
<td>The Master Problem</td>
</tr>
<tr>
<td>4.4.2</td>
<td>The Pricing Problem</td>
</tr>
<tr>
<td>4.4.3</td>
<td>The Algorithm</td>
</tr>
<tr>
<td>4.5</td>
<td>Implementation Details</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Test Problems</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Computational Platform</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Computational Results</td>
</tr>
<tr>
<td>4.6</td>
<td>Summary and Conclusions</td>
</tr>
<tr>
<td>5</td>
<td>SOLVING THE CURFEW PLANNING PROBLEM</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>Problem Description</td>
</tr>
<tr>
<td>5.3</td>
<td>Literature Review</td>
</tr>
<tr>
<td>5.4</td>
<td>Time-Space Network Formulation (TSNF)</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Mathematical Model</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Computational Analysis</td>
</tr>
<tr>
<td>5.5</td>
<td>Duty-Generation Model (DGM)</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Duty-Generation Phase 1: Variable Reduction</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Duty-Generation Phase 2: Project Scheduling</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Duty-Generation Phase 3: Crew-Scheduling</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Computational Analysis</td>
</tr>
<tr>
<td>5.6</td>
<td>Column-Generation Model (CGM)</td>
</tr>
<tr>
<td>5.6.1</td>
<td>The Master Problem (MP)</td>
</tr>
<tr>
<td>5.6.2</td>
<td>The Pricing Problem</td>
</tr>
</tbody>
</table>
5.6.3 Computational Analysis ........................................ 139
5.7 Decomposition-Based Duty Generation Model ...................... 141
5.8 Computational Results ............................................... 143
5.9 Conclusions .............................................................. 147

6 GENERAL CONCLUSION AND FUTURE RESEARCH ............... 149

REFERENCES ................................................................. 152

BIOGRAPHICAL SKETCH .................................................. 158
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Examples of railcars and candidate patterns</td>
<td>24</td>
</tr>
<tr>
<td>2-2</td>
<td>Gaps at different time intervals by IPAM.</td>
<td>43</td>
</tr>
<tr>
<td>2-3</td>
<td>Comparative analysis of heuristics.</td>
<td>43</td>
</tr>
<tr>
<td>2-4</td>
<td>Development of hybrid approach (time in seconds).</td>
<td>44</td>
</tr>
<tr>
<td>3-1</td>
<td>Comparative analysis of exact SDMCC algorithms: small instances.</td>
<td>77</td>
</tr>
<tr>
<td>3-2</td>
<td>Comparative analysis of exact SDMCC algorithms: medium instances</td>
<td>78</td>
</tr>
<tr>
<td>3-3</td>
<td>Comparative analysis of exact SDMCC algorithms: large instances</td>
<td>79</td>
</tr>
<tr>
<td>3-4</td>
<td>Comparative analysis of exact SDNCC algorithms: small instances</td>
<td>81</td>
</tr>
<tr>
<td>3-5</td>
<td>Comparative analysis of exact SDNCC algorithms: medium instances</td>
<td>82</td>
</tr>
<tr>
<td>3-6</td>
<td>Comparative analysis of exact SDNCC algorithms: large instances</td>
<td>83</td>
</tr>
<tr>
<td>3-7</td>
<td>Comparative analysis of heuristic SDMCC algorithms</td>
<td>86</td>
</tr>
<tr>
<td>3-8</td>
<td>Comparative analysis of heuristic SDNCC algorithms</td>
<td>87</td>
</tr>
<tr>
<td>4-1</td>
<td>Constraints and associated dual values for the Master Problem</td>
<td>100</td>
</tr>
<tr>
<td>4-2</td>
<td>Results on the LRP benchmark instances</td>
<td>110</td>
</tr>
<tr>
<td>4-3</td>
<td>Comparison on Perl’s benchmark problems</td>
<td>111</td>
</tr>
<tr>
<td>4-4</td>
<td>Comparison on benchmark problems that are compiled by Barreto [11]</td>
<td>112</td>
</tr>
<tr>
<td>5-1</td>
<td>Details of real-life instances</td>
<td>143</td>
</tr>
<tr>
<td>5-2</td>
<td>Improvement in number of violations for 2007</td>
<td>145</td>
</tr>
<tr>
<td>5-3</td>
<td>Improvement in number of violations for 2007</td>
<td>145</td>
</tr>
<tr>
<td>5-4</td>
<td>Comparison of algorithms: optimality gap (small instances)</td>
<td>147</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2-1</td>
<td>Simplified model of a railroad terminal.</td>
<td>22</td>
</tr>
<tr>
<td>2-2</td>
<td>Patterns for a railcar.</td>
<td>25</td>
</tr>
<tr>
<td>2-3</td>
<td>Flow of units at a railroad terminal.</td>
<td>26</td>
</tr>
<tr>
<td>2-4</td>
<td>50% hitch utilization.</td>
<td>27</td>
</tr>
<tr>
<td>2-5</td>
<td>100% hitch utilization.</td>
<td>27</td>
</tr>
<tr>
<td>2-6</td>
<td>Network for LPP.</td>
<td>29</td>
</tr>
<tr>
<td>2-7</td>
<td>Nodes creation in the improvement graph</td>
<td>37</td>
</tr>
<tr>
<td>3-1</td>
<td>Illustrating subset-disjoint minimum cost cycle.</td>
<td>49</td>
</tr>
<tr>
<td>3-2</td>
<td>Illustrating subset-disjoint negative cost cycle.</td>
<td>50</td>
</tr>
<tr>
<td>4-1</td>
<td>Example data for the LRP</td>
<td>91</td>
</tr>
<tr>
<td>4-2</td>
<td>Dynamic programming algorithm for ESPPRC.</td>
<td>102</td>
</tr>
<tr>
<td>4-3</td>
<td>Column generation algorithm</td>
<td>106</td>
</tr>
<tr>
<td>5-1</td>
<td>Space-Time network for the curfew planning problem.</td>
<td>123</td>
</tr>
<tr>
<td>5-2</td>
<td>Variable matrix D[p] for project p</td>
<td>129</td>
</tr>
<tr>
<td>5-3</td>
<td>Dynamic programming algorithm for ESPPRC.</td>
<td>140</td>
</tr>
<tr>
<td>5-4</td>
<td>Algorithm framework of the column-generation procedure.</td>
<td>141</td>
</tr>
</tbody>
</table>
COMBINATORIAL APPROACHES TO SOLVE SCHEDULING PROBLEMS

By

Ashish Kumar Nemani

December 2009

In this dissertation, we discuss classical scheduling problems especially in transportation industries. All of these problems come from real life applications and are very critical from the academic as well as the financial point of view. Being NP-hard problems there doesn’t exist any well-defined algorithm which can solve them efficiently with good running time, and thus rules of thumbs are still being followed in practice, with very myopic use of optimization procedures. These decisions are usually worth billions of dollars per year and even a slight improvement will have a significant economic impact. Through efforts described in this document, we try to develop some holistic approaches along with heuristics, to get efficient and effective results for these decision problems. This proposed work has the potential of implementation in commercial grade software. We first suggest some hybrid approaches to solve the intermodal load planning problem, which generate very effective solutions within minutes. In our second problem, subset-disjoint minimum cost cycle problem, we suggest several exact and heuristic approaches to find the minimum cost cycles which contains at most one node from any subset. These problems occur very often as a subproblem of other other combinatorial problems. In the third problem, location routing problem, we suggest column generation algorithm and show its effectiveness by doing experiments with benchmark problems. In our last problem, which is very critical in all railway industries, we propose several models based on based on mixed integer programming, heuristics,
and other hybrid approaches. These algorithms show a significant improvement over the current practices.
CHAPTER 1
INTRODUCTION

The level of complexity in today’s world is unprecedented. Most of the real life problems of industrial applications are very large scale optimization problems. The scheduling problems are the most versatile among these, occurring almost everywhere and most importantly in the transportation industries. The development of the global economy, advancement in infrastructure, and growth in technology have resulted in exponential rise in production and freight transportation. The concepts of intermodalism and containerization have increased the transportation efficiency to the point of diminishing returns for capacity extension. In this scenario further improvement is possible only with better optimization routines. The railway mode of the intermodal transportation contains a pool of decision problems such as locomotive planning, train scheduling, facility location of the storing lot, load scheduling, curfew planning, and resource scheduling.

In this dissertation we consider the scheduling problems, especially in the transportation industries. All of these problems come from real life applications and are very critical from the academic as well as the financial point of view. Being NP-hard problems there doesn’t exist any well-defined algorithm which can solve them efficiently with good running time, and thus rules of thumbs are still being followed in practice, with very myopic use of optimization procedures. These decisions are usually worth billions of dollars per year and even a slight improvement will have a significant economic impact. Through efforts described in this document, we try to develop some holistic approaches along with heuristics, to get efficient and effective results for these decision problems. This proposed work has the potential of implementation in commercial grade software.

In a railroad industry, most of the processes have inherent scheduling problems arising from the development of yearly maintenance schedules of the railway infrastructure
to the load scheduling of containers and trailers on the railcars of a train. The allotment of resources to the tasks based on their efficiency is another well researched combinatorial problem which finds application in the railroad (for e.g., distribution of material handling equipment to transport the containers to and from the railway platform to the storage area).

The intermodal transportation is the movement of freight from its source to destination through several modes of transportation such as rail, truck, and ship. The change in the transportation modes through the journey needs large material handling requirement which led to the genesis of the concept of containerization. The containers and trailers became the basic entities of transportation. These exist in several sizes, shapes and with specific handling requirements. In the railway mode of transportation these have to be loaded on railcars using handling equipments. These railcars and other equipment (cranes, chassis etc.) vary in capacity and have to follow several regulations to carry the containers and trailers. These issues create several decision issues such as the matching of railcars and containers, cranes and containers, chassis and trailers. Herein, we focus on the matching of railcars, containers, and trailers.

The matching of trailers and containers with railcars, and determining their positions (configuration), is the load scheduling problem. Based on its design, a railcar can carry the containers in several configurations such as single stacked, double stacked, etc. The number of these configurations, also called patterns, is exponential and hence, a decision support tool is required for the optimum loading. Second factor, which is very critical for the fuel efficiency of a train while determining the configurations, is the gap between the units of two railcars. The air trapped in the gap reduces the aerodynamic efficiency of a train and can increase the fuel consumption by as much as 20%. Moreover, it further increases the complexity of the problem to another dimension.

Herein, we propose a hybrid approach to solve the load scheduling problem. We formulate the problem as an Integer Programming Assignment Model (IPAM) and
generate an average quality solution. We improve this solution by Very Large Scale Neighborhood (VLSN) heuristics. We have also developed two heuristics which differ in the neighborhood size. The Single-Unit Single-Exchange Neighborhood (SUSEN) is very fast but the improvement opportunities are limited. The improvement chance is increased by allowing multiple units exchanges through Multiple-Unit Multi-Exchange Neighborhood (MUMEN) search. The results obtained are very promising and can be implemented for real life applications. In some instances the solution quality is not very good, and we plan to improve the algorithm by better mixing of IPAM and heuristics to make the algorithm more robust in the near future.

The subset disjoint minimum cost cycle (SDMCC) detection problems are encountered as a part of most of the routing and scheduling problems. The term "cost" is usually replaced by "weight" or "length". In a directed graph $G = (N, A)$, where $N$ is the set of nodes, and $A$ is the set of arcs, the minimum cost cycle algorithm finds a cycle with minimum cost/length/weight which visits each node at most once. In the case of subset disjoint graphs, the nodes of the network are dividing among subsets $S$. The subset disjoint minimum cost cycle algorithms find a cycle with minimum cost/length/weight which visits each subset (through any node) at most once. The subset disjoint negative cycle problem is a special case of subset disjoint minimum cycle problems in which minimum cost must be negative. This special requirement supplements the algorithms with unique characteristics which are utilized to speed-up the algorithms. We have proposes four exact algorithms to find subset disjoint minimum cost cycle, along with their modifications which are used to detect only the negative cycles. We thereafter propose three heuristics which control some parameters of these exact algorithms, and generate minimum or negative cycles very quickly. We finally perform extensive computational experiments and compare all proposed algorithms.

The location routing problem (LRP) is discussed in the fourth chapter of this dissertation. It is a combination of two classical optimization problems: (i) facility
location problem; and (ii) vehicle routing problem. Both the constituting problems are NP-Hard and therefore, the combined problem is also a hard problem. Since, the location of facilities in the vehicle routing problem has a great effect in the total cost of the distribution system; these should be solved within the same optimization framework. Sometimes, it is argued that the horizon of two constituent problems being quite different: facility location long term; vehicle routing: short term), they should not be combined. But this argument is valid is dependent on particular scenarios. Several applications such as blood bank location, newspaper location, post box location, waste collection, goods distribution, and parcel delivery; must have both the decisions combined. We have a vast amount of literature devoted independently to each of the constituent problems, but the LRP is still a center of attraction for researchers.

The LRP consists of selecting a subset of locations among given potential facility locations and determining the vehicle routes to visit all customers from the selected facility locations. The objective is to minimize the sum of fixed costs of setting up the facilities, variable costs of production at each location, and delivery costs such that the demand of each customer is satisfied without violating the facilities or vehicle’s capacity restrictions. In this dissertation, we first introduce the mixed integer programming formulation for the LRP. Since, the complexity of problem renders the standard optimization using MIP ineffective, we propose a column generation algorithm. To explain this algorithm we first introduce the concept of master problem and sub problem which are two main constituent of the column generation. The master problem and sub problems are repetitively solved finally converging towards a near-optimal solution. We develop the dynamic programming algorithm for the pricing problems. These algorithms are tested on several benchmark problems. In the computational analysis, our algorithms have shown vast improvement almost all instances.

The curfew planning problem is a relatively new problem from the existing literature point of view but in practice is very old. A major railroad has its infrastructure network
spread throughout the country and it keeps growing along with the growth in economy and demand. The railway track is one such entity which needs regular maintenance along with the extension. At the start of each year the pool of tracks to be repaired is created. Based on their locations and type of repairs, these pools are categorized into projects. Some projects are done on the rails, called rail-projects, and some on the ties joining rails, called tie projects. There are four groups of crew, specialized in different jobs, for the completion of projects. Most of the projects when being repaired or replaced cause a disruption in the schedule of the trains as these block the regular path and that is why the region is called under curfew. The process of preparing a schedule to minimize the disruptions is known as curfew planning.

The railway network is usually classified into several subdivisions. The excessive weather conditions put restrictions on the time-windows for repairs and replacements. There are some subdivisions which complement each other and cannot be put into maintenance at the same time. The railroads usually have several service corridors each having a group of subdivisions handling most of the traffic. It is important to keep the regular flow of traffic and hence only one of these subdivisions is released at a time. After completing a project, crews travel during the weekend to another project location which exists within a certain distance limit. All these restrictions make the problem combinatorial in nature.

We have proposed several approaches to solve the problem. We first model the problem on the time-space network but the enormity of the problem renders this approach ineffective. We introduce some variable reduction schemes by relaxing the distance limits but the solution quality degrades. These results led us to our second approach called the column generation. We decompose the problem into a master problem and sub-problem. Master problem takes several feasible potential schedules of each crew and computes the best combinations as the output. The sub-problem generates feasible schedule with respect to the time-windows, precedence, and
distance limit. This approach had moderate success level and emphasized the need of project crashing. Decreasing the duration of a job by putting two teams together is called project crashing. Inclusion of project crashing into the column generation model makes the sub-problem more complex. Finally, this led to our third approach called "duty generation model (DGM)". We solve the problem in three phases, each handling a set of constraints. First phase handles the time-window constraints, crashing opportunities, and jamboree restrictions. Second phase tackles all other constraints excluding the distance limit restriction. In the third phase we try to minimize the distance violations using a minimum cost flow model. This model is very effective with running time and solution quality, if we ignore the distance constraints. The second phase of the model restricts the third phase, resulting in large number of distance-violations. The computational results of the DGM encouraged us to develop a decomposition based algorithm, which we call decomposition-based duty generation algorithm. It first uses the DGM to generate a partition of projects among three groups. In the second phase, we solve each partition independently, and finally in the third phase we merge the solutions of each phase into one global solution. The decomposition-based DGM has shown significant improvement over other approaches as well as industry solution.

The ensuing document is organized as follows. The Chapter 2 deals with the intermodal load scheduling problem. Chapter 4 discusses the location routing problem. Chapter 3 deals with the subset disjoint minimum or negative cycle problems. Chapter 5 deals with the curfew planning algorithm. Finally, we give a general conclusion and future work in the 6th chapter.
CHAPTER 2
LOAD PLANNING PROBLEM AT AN INTERMODAL RAILROAD TERMINAL

2.1 Introduction

Intermodal transportation is the shipment of containerized cargo using more than one mode of transportation, which is usually ship, truck, or rail. From the seaports to the railway terminals, there are a series of operations that regulate the flows of several entities such as railcars, containers, trailers, handling equipment, etc. [54]. Intermodal trains’ transport goods are stored in containers or trailers and then loaded on the railcars. Determining the optimal position of these containers and trailers on the railcars is a very critical decision factor, because it affects the economy of the trains. Our research is focused on this aspect of the railway-mode and is referred to here as the load planning problem.

The development in the railway-mode has revolutionized the growth of intermodal transportation, and it is currently among one of the fastest-growing businesses in the U.S. rail industry. In the last decade, the growth rate in transport through this mode has been over ten percent [32]. In 2001, the freight traffic in ton-miles by rail was 47% in comparison to trucks’ 33% and ships’ 20% [73]. Earlier research has shown that for distances beyond 310 miles, this mode of transport is more efficient and results in savings in operational costs and labor [16]. Rail transport also diverts some freight traffic from the roads, partially alleviating congestion and wear and tear on highways [55]. There also is a great scope for improvement due to more developed infrastructure, better planning, and better strategic decisions [28].

The demand trend for rail service has been very high, but the operational activity has not been able to match this surge due to inefficient techniques and other policies [73]. Several decision issues arise when planning the delivery of a container from its origin point through to its destination point, such as the time at which it should be dispatched, its position in the storage yard at the terminal, the train and the railcar on
which it should be loaded, etc. Loading containers and trailers on the railcars is a very complex process, because they have different characteristics. There are several types of containers and trailers which differ in size, shape, and possible loading configurations. These also may have special requirements such as providing refrigeration or being able to safely convey hazardous materials. Some containers can be double-stacked, while others can be put only at the bottom level of railcars capable of double-stacking. And then the railcars, too, vary in shape, size, loading capacity, and capability of meeting special requirements. In addition to optimum space utilization, aerodynamic issues also are critical in fuel efficiency measurement. Class I railroads spent over $3 billion on fuel in 2003, and the fuel cost has increased by more than 60% since 1998 [48]. Hence, even a slight improvement in the space or fuel efficiency will save millions per year. The aerodynamic factor of a train is a function of two main entities: the gap between the units (containers or trailers) loaded on adjacent railcars, and how far the gap is from the front end of the train. All of these characteristics make the problem highly combinatorial, and it can be proven NP-Complete by reduction from a generalized bin packing problem. Different variations of the problem have been extensively studied in the literature, and we briefly discuss those which are closest to the load planning problem.

One of the first models was developed by Feo and Velarde [32]. Their paper solves a relaxed load planning problem, which considers only trailers and assumes that a railcar can carry at most two trailers. It gives an integer programming formulation of the problem based on the set covering approach. Powell and Carvalho [66] are the first contributors who consider the space and time network structure of this problem. They introduce the model for optimizing the flow of entities over several terminals in a planning horizon, which respects the technical constraints related to assigning the units on railcars. However, they don’t consider the aerodynamic issues. They formulate the problem based on a logistic queuing network, called LQN [67]. Newman and Yano [59] consider a variant of the problem by combining the train scheduling
with the load planning process. They model the problem on a multiple-fixed-cost, multicommodity network structure and formulate it as an integer program. Corry and Kozan [24] propose a multi-objective integer programming model, which develops a load plan to optimize the total handling time of loading and the balanced mass distribution on the train. The mass distribution of the train should be biased toward the front to reduce the wear on the braking mechanism. Bostel and Dejax [17] address the problem of optimization of container loading on trains in a rail-rail transshipment shunting yard to minimize the transfer within the yard. The aerodynamic issues were first captured and modeled by Lai and Barkan [48], [49]. They discuss several options for improving the aerodynamic efficiency and fuel consumption of the trains. Their extensive study helped us to identify the critical factors associated with an intermodal freight train's efficiency. The most recent paper by Lai, Barkan, and Onal [50] formulates the problem as an integer programming model, whose objective is to minimize the fuel consumption. We generalized their approach and added other dimensions to make it closer to a real-life scenario.

To summarize, the algorithms developed thus far for solving the LPP have been very helpful to us in understanding all related issues and in the development of a unique model that can address all relevant factors and can be extended to other dimensions. We initially formulate the problem as an MIP and try to solve it using exact algorithms. This model first creates a pattern set (arrangement of units on railcars) and then uses it in the next phase for constructing the network. It improves the flexibility, as some patterns that are not covered by rules can be added from the historical data. We also develop two very large-scale neighborhood (VLSN) heuristics utilizing tabu search that consider the global problem and generate excellent-quality solutions within 30 seconds. VLSN search algorithms are generally applicable to problems which can be represented as partition problems [3]. The LPP is not a true partition problem (see Section 2.5), so we modify the basic VLSN algorithm accordingly and extended it using the tabu search,
which makes it more robust. Finally, we develop and use a practical, very efficient hybrid approach that combines the best qualities of MIP and VLSN searches. Our hybrid algorithm solves the problem within four minutes and generates near-optimal solutions. Additionally, this approach can be applied easily to the dynamic version of the problem by making appropriate changes in the heuristic phase. Our paper is organized as follows:

1. We give a detailed description of the intermodal railway terminal and the entities (Section 2.2).

2. We explain the restrictions on assigning containers or trailers on railcars, the guiding criteria for the loading process, as well as other critical factors (Section 2.3).

3. We formulate the load planning problem as an MIP on an appropriately defined network with generalized flow and describe the formulation (Section 2.4).

4. We propose a heuristic, single-unit multi-exchange neighborhood (SUMEN) search, which generates a locally optimal solution within two seconds (Section 2.5).

5. We describe the second heuristic, multi-unit multi-exchange neighborhood (MUMEN) search, which generates near-optimal solutions in a very short time. We then improve it by combining with the tabu search (MUMENT). We then suggest a unique hybrid procedure that produces solutions better than those generated by CPLEX in more than one hour (Section 2.7).

6. We perform extensive computational investigations of the exact, heuristics, and hybrid approaches and report these results (Section 2.8). We also discuss the applicability of each approach in real-life applications.

7. We describe the flexibility of our approaches for solving different variations of the problem (Section 2.9). This includes discussing the dynamic version of the problem and suggesting how it can be solved by directly extending our approach.

8. Finally, we present our conclusions and suggest directions for future research (Section 2.10).

### 2.2 Definitions and Background

All the modes of intermodal transportation can be collectively considered a system and each mode its subsystem. The railway subsystem consists of terminals, tracks,
locomotives and railcars, intermediate hubs, rail yards, containers, trailers, road railers, and material-handling equipment (cranes, chassis). Containers and trailers often are considered the same entity and are referred to as “units.” A very simplified picture of a terminal is shown in Figure 2-1. We briefly will discuss the main entities, units, railcars and handling equipment that play significant roles in all the decision problems at this subsystem.

Figure 2-1. Simplified model of a railroad terminal.

Containers and trailers are used to transport freight in the intermodal transportation. The difference between these two is that the trailers have a set of wheels used to drive on the road, while the containers need to be transferred onto suitably sized trucks. Containers and trailers vary in their characteristics depending on their types. The most common container sizes used in international commerce are 20 ft., 28 ft., 40 ft., and 48 ft. Other sizes are: 10 ft., used primarily in place Europe and by the military services; 24 ft.; 45 ft.; 53 ft.; and 56 ft., usually used at the domestic level. Typical container
heights are 8 ft., 6 inches, and 9 ft., 6 inches, while some containers are less than 8 feet and are used for triple-stacked shipments of automobiles. The standard width of containers used in international transport is 8 feet. [68]. Trailers also have a standard width (8 ft., 6 inches) and height (9 ft., 6 inches), and a length varying from 28 to 57 feet. Some containers have special features and are grouped as special-purpose units used for carrying temperature-controlled cargo, liquids, gases, automobiles, and livestock.

In rail-transport, containers and trailers are placed on the railcars using handling equipment.

A train generally consists of 60-85 railcars and a few locomotives, and its capacity is given either in terms of maximum length or weight. There are several types of railcars such as double-stacks, spine-and-skeleton container cars, skeleton trailer cars, piggyback trailers, and road railers. Table 2-1 shows some examples of railcars and their characteristics. Railcars come in several configurations based on the number of hitches on them and their relative positions on the platform. These hitches define the slots where containers/trailers may be loaded. The number of hitches on the railcars varies from one (frontrunners) to seven (spine cars). For trailers, there are some special arrangements for the holding and safety of their wheels. Road railers don’t require a railcar and attachments are made directly to their backs. The conveyance of trailers and containers on railcars is referred to as trailer-on-flatcar (TOFC) and container-on-flatcar (COFC). The transport using COFC is increasing quite rapidly compared to TOFC. In 1996, the COFC market share accounted for only 77 percent of the total intermodal volume, while in 2001, it accounted for more than 92 percent [63].

The transport of containers and trailers within a yard and their loading is performed using handling equipment such as dockside cranes, stackers, packers, front lift trucks, side loaders, straddle carriers, stacking gantry cranes, and chassis, etc. At intermodal yards, a chassis is a trailer designed specifically to haul containers. For simplicity, this
equipment is divided into two categories – machines that approach the units from the side and machines that move over the top of, or straddle them [55].

Table 2-1. Examples of railcars and candidate patterns

<table>
<thead>
<tr>
<th>Car</th>
<th>Length</th>
<th>Containers</th>
<th>Trailers</th>
</tr>
</thead>
<tbody>
<tr>
<td>89-ft flat</td>
<td>89'</td>
<td>20'/40'/45'</td>
<td>2 45'/3 28'</td>
</tr>
<tr>
<td>85-ft flat</td>
<td>85'</td>
<td>4 20'/2 40'</td>
<td></td>
</tr>
<tr>
<td>60-ft flat</td>
<td>60'</td>
<td>3 20'/20'+40'</td>
<td></td>
</tr>
<tr>
<td>“Twin-45” flat</td>
<td>89'</td>
<td></td>
<td>2 45'</td>
</tr>
<tr>
<td>“Front Runner” flat</td>
<td>50’</td>
<td>1 40'/1 45'/1 48’</td>
<td></td>
</tr>
<tr>
<td>“Long Runner” flat</td>
<td>Joining 2 89’ flatcars</td>
<td>4 45'/3 48'/3 53'/3 57’</td>
<td></td>
</tr>
<tr>
<td>Arc 5</td>
<td>5 unit car</td>
<td>40'/45'/48’</td>
<td></td>
</tr>
<tr>
<td>Arc 3</td>
<td>3 unit car</td>
<td>40'/45’</td>
<td></td>
</tr>
<tr>
<td>Trailer Railer</td>
<td></td>
<td>28’-60’</td>
<td>28’-60’</td>
</tr>
</tbody>
</table>

2.3 Problem Description

There are two major issues in transferring the units between the origin and destination terminal-pair. First is the blocking problem, where a train has to be built by grouping railcars positioned on different tracks of the terminal. Second is the load planning problem (LPP), which deals with the optimal loading of the units on these railcars. Each of the issues affects the output of the other, but the LPP can be considered as the subproblem and solved independently for each block. As shown in Figure 2-1, there are several tracks with railcars and units of different destinations, which are delivered and stored in yards. As the new units are delivered, the load plan is revised with the modified set of railcars and units. In a railroad, this dynamic decision-making process is currently performed using rules of thumb by experienced persons who coordinate the material-handling operations. Some railroads use a comprehensive decision support system whose primary function is to regulate the information flow between different channels of the firm rather than to optimize the loading process. They try to assign the units to the railcars after considering several restrictions that allow only a subset of arrangements. We refer to these arrangements as patterns.
A pattern is a collection of units in a certain arrangement that can be assigned to a railcar. The number of patterns to which a railcar can be assigned is quite large, although only a few types of units can be assigned to a railcar, which is under several restrictions that we discuss later. Many guidelines must be considered before generating the patterns for a railcar. Some examples of patterns are shown in Figure 2-2. Next, we briefly discuss the operations which lead to the formation and loading of patterns on the railcars.

2.3.1 Basic Operations

The containers and trailers are delivered from the local customers to a terminal and then stored at a specified place in the intermodal yard. Later they are brought to the platform and loaded on an outbound train using the handling equipment. Similarly, the units from the inbound trains are unloaded and brought into the yard for dispatching to the customers or for reloading onto different outbound trains (see Figure 2-3). A load plan should optimize the loading and unloading processes together. When a train arrives at the platform, generally, unloading and loading start simultaneously. Trucks carrying outbound containers are directed to the railcars on which the containers are to be loaded. If there is space on that railcar, the container is loaded directly on it; otherwise, it is placed on the ground and loaded after the railcar is emptied. This process is called double-handling, as one additional move of the container is required.

Loading the units is a dynamic process, and the loading plan is revised as new events occur, such as the arrival of new units. In the revised plan, a container that already has been put on the ground may be assigned to a different railcar. This reassignment introduces another movement of the container.
After the units are loaded on the railcars, they are secured using fastening devices called pins that are located on the railcars. The configuration of pins in the new arrangement of units on the railcars impacts the loading efficiency.

2.3.2 General Rules of Loading Units on Railcars

AAR safety regulations constrain the freedom of loading [1]. Some of the constraints are listed below:

1. There should be a minimum clearance, about 10 inches, between the trailers on the adjacent railcars.
2. There should be a minimum working distance of at least two inches between any two adjacent trailers to facilitate smooth loading and unloading operations.
3. A trailer may not overhang the strikers by more than five inches on each end of a railcar.
4. The width of a trailer must not exceed the width of the railcar to avoid damage to the tires.
5. The weight distribution on the train should be biased toward the front end.
6. While stacking, empty containers cannot be put at the bottom.
7. Smaller containers (26 ft., 28 ft.) generally carry heavy commodities and should be put at the bottom level in double-stacking to avoid wear and tear as well as accidents.

There are other common restrictions, such as a trailer cannot be double-stacked. To support the double-stacking, containers have pins only at the corner. This eliminates the possibility of putting a small container on a comparatively large one and vice-versa.

Special purpose units must be handled more carefully. Refrigerated containers and trailers need more space for proper functioning. Units containing hazardous materials
require extra safety arrangements. All of these restrictions make the problem very complex, and so utilizing the full capacity of the railcars becomes a research topic.

2.3.3 Hitch Utilization

Railcars have different capacities for carrying the units depending upon their characteristics, such as size, number of hitches, double-stack feasibility, etc. Units are loaded on the railcars between the hitches. Hitch utilization is defined as the percentage of loaded hitches with respect to the total number of hitches pulled [32]. Figures 2-4 and 2-5, respectively, give examples of poor and good hitch utilization on a double-stack railcar. This is one of the efficiency measurement factors of a loaded train. Higher hitch utilization significantly reduces the cost of pulling an outbound train, as it reduces the number of railcars required. Also, the denser arrangement of units entraps a lesser volume of air that is dragged and hence lowers the fuel consumption (see details in Section 2.3.4).

Figure 2-4. 50% hitch utilization.

Figure 2-5. 100% hitch utilization.

Current hitch utilization in the United States ranges between 80 and 95 percent. A study at Conrail, one of the leading rail companies in the United States, has shown that a one percent improvement in hitch utilization could save 450,000 dollars per year at each of its fifteen yards [32].

2.3.4 Aerodynamic Efficiency

The second attribute used to assess the loading pattern of an outbound train is its aerodynamic efficiency. This factor becomes even more critical because intermodal
trains run at high speeds. Many experiments have been performed to learn the exact cause of the aerodynamic drag, and it was found that the gap length between units, the position of the gap in the train, and the yaw angle are three important factors affecting it [25]. **Yaw angle is the angle between a train’s heading and its actual direction of travel or course.** This is an important factor at a particular point, but in the global scenario it doesn’t play any active role. The yaw angle changes as soon as the train changes its course and neutralizes the corresponding factor. A larger gap length between units results in higher entrapped air volume and consequently poor aerodynamic efficiency. The experiments also suggest that the first locomotive experiences the highest drag due to the headwind impact; as the train moves farther, this drag declines until the tenth unit, and thereafter it becomes constant. This suggests that if unit and railcar combinations are such that the gap cannot be avoided, then the gap should be at the rear end of the train.

The relationship between the aerodynamic resistance and the unit’s position in a train is given as follows [25]:

\[
C_D A (\text{ft}^2) = 14.85824 e^{-0.29308k} + 9.86549 e^{-0.00007k} + 10.66914
\]

where \( k \) is the railcar’s position in the train and \( CDA \) is the drag area representing aerodynamic resistance. An **adjustment factor** is associated with each gap by dividing the drag area of that unit by that of the 100th unit [48]. The adjustment factor at a place is multiplied with the length of the gap between two consecutive units at that place to find the adjusted gap length in order to compute the drag. The **adjusted gap length is defined as the gap length weighted by the position-in-train effect** [50].

### 2.4 Model Formulation

The problem of assigning the units on a given set of railcars is itself a challenging problem because of its combinatorial nature. In this paper, we solve the static problem where all of the units to be assigned are known in advance. (In Section 2.9 we discuss
extending the model to include the dynamic factor of the problem, described previously.)

We formulate this problem as the integer program, and the underlying network is shown in Figure 2-6. It contains three sets of nodes, described as i, ii, and iii. (i) The first set corresponds to the units grouped into families. All the units of a family are of the same shape, weight, and dimensions, but they may differ in revenue earned or penalty incurred if delayed. (ii) The second set belongs to all the feasible sets of patterns, and (iii) the third set belongs to all the railcars.

The arc between a unit-node and a pattern-node shows that the unit can be assigned in that pattern at a certain cost. An arc exists between a pattern-node and a railcar-node if the units can be assigned on the railcar in that pattern. Due to the presence of these two sets of arcs, the flow through the pattern-nodes is conserved and hence, the network is a generalized flow network [5].

![Figure 2-6. Network for LPP.](image)

The following notation is used to describe the mixed integer programming formulation of the load planning problem (LPP). The proposed model will be denoted as the integer programming assignment model (IPAM). **Parameters:**

- \( B \): set of all the units.
Decision Variables:

\[ x_{ij} = 1 \text{ if unit } l \text{ is loaded in pattern } j; \ 0 \text{ otherwise.} \]

\[ y_{jk} = 1 \text{ if railcar } k \text{ is assigned in pattern } j; \ 0 \text{ otherwise.} \]

\[ z_k = 1 \text{ if railcar } k \text{ or railcar } k-1 \text{ is in double-stacked pattern; } 0 \text{ otherwise.} \]

\[ s = 1 \text{ if any unit if left on ground; } 0 \text{ otherwise.} \]

The formulation follows.

Objective Function:

\[
\text{Maximize } \sum_j \sum_l r_l x_{lj} - \sum_k \sum_j c_{jk} y_{jk} - \frac{A_k}{2} \left( l_{j1} + l_{j2} \right) * y_{j1} - \\
\sum_{k=2}^n \frac{A_k}{2} \left[ \left( \sum_j l_{jk-1} y_{jk-1} + \sum_j l_{jk1} y_{jk} \right) + \left( \sum_j l_{jk-1} y_{jk-1} + \sum_j l_{jk2} y_{jk} \right) * z_k \right]
\]
Subject to:

\[ \sum_{l \in B} x_{lj} - \sum_{k} a_{ij} y_{jk} = 0 \quad \forall i \in I, j \in P \quad (2-1) \]

\[ \sum_{j} \alpha_{jk} y_{jk} \leq 1 \quad \forall k \in K \quad (2-2) \]

\[ \sum_{j} x_{lj} \leq 1 \quad \forall l \in B \quad (2-3) \]

\[ \sum_{j} w_{j} y_{jk} \leq W_k \quad \forall k \in K \quad (2-4) \]

\[ \sum_{j} \sum_{k} w_{j} y_{jk} \leq W \quad (2-5) \]

\[ 1 - \sum_{j} x_{lj} \leq s \quad \forall l \in B \quad (2-6) \]

\[ \sum_{j} y_{jk} \geq s \quad \forall k \in K \quad (2-7) \]

\[ 2z_k - \sum_{j} \beta_{jk} y_{jk} - \sum_{j} \beta_{jk-1} y_{jk-1} \geq 0 \quad \forall k \in K \quad (2-8) \]

\[ z_k - \sum_{j} \beta_{jk} y_{jk} - \sum_{j} \beta_{jk-1} y_{jk-1} \leq 0 \quad \forall k \in K \quad (2-9) \]

\[ X, Y, Z, s = \{0, 1\} \quad (2-10) \]

The objective function consists of the revenue generated by transporting the units from the origin to the destination, the cost of loading a car in a particular pattern, and the drag caused by the gap between units loaded on the railcars. The revenue can be replaced by a penalty function for not assigning a unit on a particular day. The last term in the objective function is nonlinear, but it can be converted easily into linear terms using standard methods when both of the variables involved are binary [58].

Constraint 2–1 is the mass balance for the inflow and outflow of containers and trailers at the pattern nodes. Constraints 2–2 and 2–3 enforce the condition that a railcar can be assigned in only one pattern and a unit can be loaded in a single pattern and thus to a single railcar, respectively. Constraints 2–4 and 2–5 ensure that the
weight-carrying capacity of each railcar as well as that of the train is not exceeded. Constraints 2–6 and 2–7 maintain the condition that if a container has been left unloaded, then none of the railcars remain unassigned. In the example here, we assume that all of the railcars are flexible enough to accommodate any type of unit. Constraints 2–8 and 2–9 give the relationship between \( y \) and \( z \) variables. Finally, nonnegativity and integrality constraints are imposed on all of the decision variables. The length restriction of the railcars is taken into consideration while building the patterns. The weight constraints also can be considered at that stage, and it has been included in the formulation only to make it more flexible.

The IPAM model captures all relevant issues and gives a detailed view of the complete problem. In an intermodal train, there are on average 60-80 railcars and 200-300 units. We consider that there are four types of railcars and that all of them are capable of being double-stacked. We generate 200-300 patterns for these railcars-units combinations. The number of variables is \( O(|P|^*|B|+|P|^*|K|) \), which on average is 80,000, and the total number of constraints is around 5,000. The IPAM model is very flexible in incorporating additional features, because it considers patterns as a separate entity. We perform the computational testing on several instances created randomly (see Section 2.8 for details) and solve them using CPLEX 11.2. We find that for the instances where the available capacity (railcars) is less than the demand (units), IPAM produces an optimal solution within five minutes. However, the solution quality deteriorates as the capacity and demand become comparable. In real-life applications, railroad planners have to prepare plans for several trains each day. Furthermore, once a solution is available, they usually need to make several changes which cannot be captured in the model. Planners also may want to change the priority of units and railcars based upon the output of first model. To make all of these changes, a fast response time is required, and a five-minute limit is practical based on the planner’s experience. These observations guided in developing heuristics that can address all of the issues.
discussed earlier and can generate acceptable solutions within the allotted time. The algorithm should be able to perform the sensitivity analysis effectively. Now we will discuss some heuristic approaches based on the concept of local search.

2.5 Heuristics: Very Large Scale Neighborhood Search Algorithms

For the load planning problem, since a unit can be assigned in different patterns on many railcars and a railcar can also be loaded with different types of units, the number of possible configurations is very large. The structure of the problem makes it NP-complete, which can be proven easily by reduction from the generalized bin packing problem. These characteristics of the problem lead toward heuristics for solving the specific problems.

Loading multiple units on the railcars starting at the front may seem intuitive, but it often results in very poor assignments. For example, consider the simple scenario of three 28–ft. and three 48–ft. units with a set of 85–ft. railcars. Loading all 28–ft. units on the first railcar and the remaining three 48–ft. units on three railcars would require four railcars, while coupling each 28–ft. unit with a 48–ft. unit will use only three railcars. So, a basic greedy approach doesn’t work for this problem. A general observation is that if a railcar is capable of being double-stacked, the presence of single-stacked containers signifies inefficiency in the loading.

Many heuristics have been proposed in earlier research. Feo and Velarde [32] suggest a greedy-randomized-adaptive-search-procedure (GRASP) for approaching the reduced load planning problem. For optimizing the aerodynamic efficiency of the train, Lai, Barkan, and Onal [50] propose a heuristic that first evaluates the aerodynamic efficiency of all possible pattern-car combinations for each type of railcar and then sorts them. Next, it considers the first slot of the railcar and assigns the best possible combination of unassigned units on it. The process is repeated until either all slots are filled or all units are assigned. This heuristic works well for the spine
cars (single-stacked); however, for the well cars (double-stacked), the results may be unsatisfactory.

We now propose a neighborhood search algorithm for the load planning problem (LPP) that has exhibited excellent computational results. This algorithm is an application of the very large-scale neighborhood (VLSN) search to the LPP at a railway intermodal terminal. A VLSN search algorithm is a neighborhood search algorithm, where the size of the neighborhood is very large and where we use an implicit enumeration algorithm to identify an improved neighbor. We refer the reader to the paper by Ahuja et al. [3] for an overview of VLSN search algorithms. These algorithms are designed to solve the partition problems as shown in the literature.

Our problem doesn’t fall exactly into this category due to the aerodynamic factor that is based on the gap between adjacent railcars. We handle this issue while finding the negative cycles by dynamically updating the arc costs. This modification prevents the basic structure of the algorithm from changing, and lets us consider the LPP as a special case of the partition problem, with the set of railcars and units partitioned into many subsets based on many rules discussed later in this section. A partition problem is to partition a set, say \( S \), into different subsets, say \( S_1, S_2, S_3, \ldots, S_k \), such that the cost of partition is minimum, where the cost of partition is the sum of the cost of each part. The partition problem was solved using VLSN by Thompson and Orlin [71] and Thompson and Psaraftis [72]. Ahuja et al. [6], [7] applied this approach to the capacitated minimum spanning tree problem and to the weapon target assignment problem [4]. We will present a brief overview of this approach when it is applied to the LPP.

A neighborhood search algorithm starts with a feasible solution of the optimization problem and successively improves it by replacing it with an improved neighbor until it reaches a local optimum. The quality of the locally optimal solution depends both upon the quality of the starting feasible solution and the structure of the neighborhood;
that is, how we define the neighborhood of a given solution. We will now describe the method we used to construct the starting feasible solution, followed by our neighborhood structure.

### 2.5.1 Construction Heuristics

We developed a simple but effective construction heuristic that builds a foundation for the VLSN search. It considers the set of first $p$ railcars and finds the best possible candidate patterns that can be built for each railcar using the unloaded units. The best railcar-pattern combination generates the maximum profit among the $p$ railcars; so it is chosen and fixed. The corresponding unit sets are updated and the fixed railcar is replaced by the next railcar in the standing order. This process continues with the new sets of units and railcars until all units are loaded, all railcars are fixed, or all alternatives are exhausted. We also can randomize the initial solution by choosing a random railcar for assignment instead of the best one from the set of $p$ railcars. The objective of this heuristic is to load as many units as possible without considering the aerodynamic factor that doesn’t affect the feasibility. We make it aerodynamically efficient in the improvement phase of the heuristic by allowing the desirable rearrangement of units on railcars in their neighborhood.

### 2.5.2 The Neighborhood Search Structure

We represent each feasible arrangement $(A)$ of the units on the railcars as a graph $G(N, A)$, with the units and railcars as nodes. The nodes that denote the railcars are called root nodes, and those that contain sets of units are children nodes. For any child node $g$, we define $C[g]$ as the root node and $S[g]$ as the set of units contained in the railcar of the root node $C[g]$. We suggest two multi-exchange neighborhood structures: Single Unit Multi-Exchange Neighborhood (SUMEN) search and Multiple Unit Multi-Exchange Neighborhood (MUMEN) search.

The networks in both the SUMEN and MUMEN searches contain two sets of nodes: railcar-sets and unit-sets. In SUMEN, each node in the railcar-set contains one
railcar and each node in the unit-set contains one unit, while in MUMEN the nodes in the unit-set may contain more than one unit. Therefore, the number of nodes in MUMEN is far more than those in SUMEN. The method of creating the nodes and arcs is described later in this section. For a given arrangement $A$ of units, given by the initial feasible solution, we define another arrangement $A'$ to be a neighbor of $A$, if $A'$ can be obtained from $A$ by performing a single-cycle exchange or a single-path exchange on the underlying network. We define a neighborhood of $A$ as a collection of arrangements that are neighbors of $A$. This neighborhood structure is obtained by performing cyclic and path exchanges in the improvement graph.

### 2.5.3 Improvement Graph

The improvement graph for the SUMEN and MUMEN structures is defined with respect to a feasible solution and is represented by $G^1(A)$ and $G^2(A)$, respectively. The graph $G^1(A)$ is a directed graph with the same node set as the graph $G$. There is a one-to-one correspondence between the nodes in $G$ and those in $G^1(A)$. The graph $G^2(A)$ contains a very high number of nodes (approximately 15 times) compared to $G^1(A)$. For each root node $C[g]$ in $G^2(A)$, we generate nodes for each possible combination of units contained in $S[g]$. Figure 2-7 shows the difference between the nodes of the unit-sets of the improvement graph for SUMEN and MUMEN structures.

The arc set in $G^1(A)$ (or $G^2(A)$) is defined in a manner such that each cyclic or path exchange with respect to $A$ defines a directed cycle in $G^1(A)$, and the cost of the cycle equals the cost of the corresponding exchange. A directed arc $(g, h)$ in $G^1(A)$ (or $G^2(A)$) signifies that node $g$ leaves its root node $C[g]$ and joins the root node $C[h]$, and that simultaneously node $h$ leaves the root node $C[h]$. To construct the improvement graph, we consider every pair $g$ and $h$ of nodes and add arc $(g, h)$ if and only if (i) $C[g] \neq C[h]$ and (ii) $S[h] \cup \{g\} \setminus \{h\}$ is a feasible pattern of railcar $C[j]$. We define the cost $\alpha_{gh}$ of arc $(g, h)$ as:

$$\alpha_{gh} = c(\{g\} \cup S[h] \setminus \{g\}) - c(S[h]).$$
2.5.4 Cyclic Exchanges

Let $A$ be an arrangement that denotes the initial feasible solution obtained by the construction heuristics. A cyclic exchange is defined by a sequence of nodes, $g_1 - g_2 - g_3 - \ldots - g_r - g_1$, where the nodes $g_1$, $g_2$, $g_3$, $\ldots$, $g_r$ belong to different root nodes in the initial solution. The cycle exchange $g_1 - g_2 - g_3 - \ldots - g_r - g_1$ represents the following changes:

- units in node $g_1$ move from the railcar in root node $C[g_1]$ to the railcar in root node $C[g_2]$,
- units in node $g_2$ from the railcar in node $C[g_2]$ to the railcar in root node $C[g_3]$, and so on, and finally units in node $g_r$ from the railcar in root node $C[g_r]$ to the railcar in root node $C[g_1]$.

This exchange is done only if the patterns generated after the exchange are feasible as well as profitable. For example, suppose that initially a railcar of type $q$ was loaded with two units of type $i_1$ and one unit of type $i_2$, and during the exchange the node containing a unit of type $i_1$ was replaced with the node containing a unit of type $i_2$. This exchange would be feasible only if the pattern with three units of type $i_1$ exists in the feasible pattern set of the railcar $q$.

Let $A'$ denote the new feasible arrangement of units after the cyclic exchange. We define the cost of this cyclic exchange as $c(A') - c(A)$.

$$c(A') - c(A) = c(\sum_{p=1}^r (c(\{g_{p-1}\} \cup S[g_p]) \setminus \{g_p\}) - c(S[g_p])), \text{ where } g_0 \text{ is defined as } g_r.$$
2.5.5 Path Exchanges

A unit-based path exchange represents the same changes as represented by the cyclic exchanges, except that the last node, \( gr \), does not move from the root node \( C[g_r] \) to the root node \( C[g_1] \).

Let \( A' \) denote the new feasible arrangement of units after the path exchange. We define the cost of this path exchange as \( c(A') - c(A) \).

\[
c(A') - c(A) = c(S[g_1]\{g_1\}) + \sum_{p=2}^{r-1} c(S[g_p]\{g_p\}) + c(S[g_r]) - \sum_{p=1}^{r'} c(S[g_p]).
\]

We don’t allow the cycle or path exchanges that involve two adjacent railcars to include the aerodynamic factor in the heuristic, as the drag depends on the gap between these two. Thus, these are to be considered together to make the LPP an instance of the partition problem. To analyze the effect of this restriction on solution quality, we solved the problem instances excluding the aerodynamic factor. In most cases, there was no effect of the restriction on the output, and in a very few instances it deteriorated marginally.

2.5.6 Identifying Path Exchanges

Next, we explain how to convert a path exchange with respect to \( G \) again into a subset-disjoint cycle with the same cost. This transformation requires an augmentation of the improvement graph by adding some nodes and arcs. We create a pseudonode for each railcar and an extra node called the origin node \( v \). For notational convenience, we refer to each original node \( g \) in the improvement graph as a regular node. In the improvement graph, we connect the origin node \( v \) to each unit-set’s node \( g \) using the arc \((v, g)\) and set its cost to \( c(S[g]\{g\}) - c(S[g]) \). We connect each pseudonode \( h \) to the origin node \( v \) using the arc \((h, v)\) and set its cost to 0. We also connect each unit-set’s node \( g \) to each pseudonode \( h \), if node \( g \) does not belong to the root node \( C[h] \) and the newly formed pattern remains feasible; this arc signifies that node \( g \) moves from the
railcar $C[g]$ to the railcar $C[h]$, but that no node moves out of railcar $C[h]$. Consequently, we define the cost of the arc $(i, h)$ as $\alpha_{jh} = c(S[h] \cup \{j\}) - c(S[h])$.

We now present some complexity analysis of the VLSN search algorithm. We obtain the starting feasible solution $A$ by using the greedy heuristic described in section 2.5.1, whose complexity is $O(pnm^*|I|)$. The improvement graph $G1(A)$ contains $O(n + |B|)$ nodes and $O((n + |B|)2)$ arcs, and the cost of all arcs can be computed in $O((n + |B|)2)$ time. We use a dynamic programming-based algorithm by Ahuja et al. [4] to obtain subset-disjoint cycles. This algorithm first looks for profitable two-exchanges involving two nodes only; if no profitable two-exchange is found, it looks for profitable three-exchanges involving three nodes, and so on. The algorithm either finds a profitable multi-exchange or terminates when it is unable to find a multi-exchange involving $t$ nodes (we set $t = 5$). In the former case, we improve the current solution, and in the latter case we declare the current solution to be locally optimal and stop. The running time of the dynamic programming algorithm is $O((n + |B|)2 * 2t)$ per iteration, and it is typically much faster since most cyclic exchanges found by the algorithm are simple swapping.

The VLSN search heuristics, particularly MUMEN, generated good-quality solutions within 30 seconds for most of the instances (see Section 2.8). For some instances, the gaps are high indicating that the VLSN search gets trapped in the local optima. To overcome this, we perturb the solution and allow some unprofitable exchanges. We use the tabu search concepts for these modifications as described in the upcoming section.

### 2.6 Very Large Scale Neighborhood With Tabu Search

As described in the previous section, the VLSN search heuristics, in particular the MUMEN, generated good-quality solutions within 30 seconds for most of the instances (see Section 2.8). The optimality gaps are good and can be improved further if more time is available. The nature of the constraints renders the feasible region highly scattered, and thus even the very large-scale neighborhood search can converge to a
solution away from the global optimal solution. To overcome this we also apply the tabu search \cite{35}, a memory-based popular meta-heuristic, along with some random solution perturbations. We have used short-term memory to avoid cycling and used perturbation to diversify the search. Next, we briefly describe our implementation of the tabu search for LPP.

The initial solution and neighborhood structure for the local search is captured in the VLSN search. Once the local optimum is reached, we perturb the solution and allow it to move to an inferior point. During the solution analysis of MUMEN, we notice intuitively that no railcar-subsequence has alternating double-stacked and single-stacked patterns. So, in perturbation we try to change these subsequences. We randomly choose two blocks of railcars, one single-stacked and one double-stacked, and put all loaded units in an unassigned unit-set. We reassign these units back on empty railcars but invert the single-stacked railcars to double-stacked ones and vice-versa. We reapply the MUMEN search starting from the perturbed solution but prohibit the algorithm in order to consider the same move for a certain number of iterations, called tabu tenure. For example, if railcar $k$ was assigned in pattern $p$ in the last local optimal solution, we will not assign it in the same pattern for a pre-specified number of patterns. If any of the moves in the cycle is a tabu, we call it a tabu cyclic exchange. We don’t accept the tabu move within the tenure, even if it may generate improved results. We terminate the search if the last two perturbations followed by the MUMEN search don’t include the best solution and if there have been at least five perturbations.

The solution generated by MUMEN combined with the tabu search (MUMENT search) improved the optimality gaps significantly, as shown in Table 2-3. We observed that IPAM generates very good results for instances with a very high demand and low capacity, while MUMENT generates consistent solutions for all. This observation motivated us to combine these two approaches into a hybrid, as we will now discuss.
2.7 Hybrid Approach

The exact solution approach (IPAM), described in Section 2.4, has a limited type of success. It generates an optimal solution in a very reasonable time frame for some problem instances, while it takes hours to reach an acceptable solution point for others. The MUMENT search generates good solutions in all instances but converges to a local optimum. We try to utilize the good characteristics of both approaches when developing a hybrid. For the hybrid approach, we start with the MUMENT search and feed its output as the initial solution into the IPAM. It serves as a good upper bound that helps the branch-and-bound procedure to prune the branches at an early stage. The IMAP preceded by the MUMENT search results either in improved solution quality or in shorter running time in almost all instances. We terminate the IPAM phase after three minutes, and if the solution obtained is not optimal, we apply the MUMENT search once again to its output. The IPAM phase usually generates a solution diverse from the last local optimal solution and helps MUMENT to diversify the search. The three-minute termination criterion is chosen from a computational as well as from a practical point of view. IPAM improves the solution very quickly at the start, but the improvement-slope becomes almost flat afterward; in a real-life situation, five minutes is a practical time boundary. From a busy terminal (BNSF, UPS), an average of 15-20 trains leave every day [38]. The load planning will be done for each train with the forecasted data, and the plan will have to be revised each time any unexpected event occurs. We discuss more about this approach with the computational results in the next section.

2.8 Computational Results

We implemented the IPAM, heuristics, and hybrid approaches described in the previous section and extensively tested them. We tested our algorithms on randomly generated instances. The data was generated in the following manner. We started by assuming that a train constitutes 80 railcars of four different lengths, 60 ft., 65 ft., 85 ft., and 89 ft., each capable of being double-stacked. In reality, the number varies
from 60 to 80. These railcars were sequenced randomly to form a train. The flexibility of the double-stacking in the railcar increases the number of candidate patterns and accordingly the complexity of the problem. We considered 13 types of units, seven containers, and six trailers that are generally used in intermodal transportation. Each unit generated a revenue (a random value) if assigned on a railcar or incurred a penalty if left on the ground. In each unit-family, there were 20 to 30 units of a fixed common weight. We varied the total number of available units from 320 to 180 by randomly removing some units from the mix. We performed all of our tests on a 1.6 GHz Pentium 4 processor computer with a 1 GB RAM PC. In this section, we present the results of these investigations.

To analyze the effect of the restriction on the exchanges between the units of adjacent cars, as discussed in Section 2.5, we performed some experiments on the relaxed load planning problem obtained by ignoring the aerodynamic factor. We observed that this restriction has a negligible effect on the quality of the solution. Out of the 15 instances investigated, only one generated a slightly better result. This experiment suggests that the restriction doesn’t reduce the neighborhood size significantly enough to worsen the solution.

2.8.1 Comparison of Exact Approach and Heuristics

In our first investigation, we observed the solution quality of IPAM at different time intervals. We started with a three-minute stopping criteria and then gradually increased it to one hour. Finally, we tried to find the optimal solution for all instances by letting it run for more than a day. The results are shown in Table 2-2. We observed that the solution quality curve is sharp in the first five minutes, but its slope keeps decreasing as time passes, therefore justifying our termination criterion of three minutes in the hybrid approach. Next, we compared the solution quality of different heuristics: SUMEN, MUMEN, and MUMENT in Table 2-3. We observed that although the SUMEN search
converges to a local optimal solution very fast, its quality is unacceptable so we didn’t include it in further discussions.

Table 2-2. Gaps at different time intervals by IPAM.

<table>
<thead>
<tr>
<th># of Railcars</th>
<th># of Units</th>
<th>3 mins</th>
<th>5 mins</th>
<th>6 mins</th>
<th>10 mins</th>
<th>60 mins</th>
<th>Optimality Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>320</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>1.6 mins</td>
</tr>
<tr>
<td>310</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>2.6 mins</td>
</tr>
<tr>
<td>300</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>5.3 mins</td>
</tr>
<tr>
<td>290</td>
<td>0.03%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>4.4 mins</td>
</tr>
<tr>
<td>280</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>3.4 mins</td>
</tr>
<tr>
<td>270</td>
<td>1.29%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>5.9 mins</td>
</tr>
<tr>
<td>260</td>
<td>0.21%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>4.6 mins</td>
</tr>
<tr>
<td>250</td>
<td>No Sol.</td>
<td>9.40%</td>
<td>9.40%</td>
<td>9.40%</td>
<td>0.00%</td>
<td></td>
<td>31.4 mins</td>
</tr>
<tr>
<td>240</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>No Sol.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-3. Comparative analysis of heuristics.

<table>
<thead>
<tr>
<th># of Railcars</th>
<th># of Units</th>
<th>SUMEN</th>
<th>Gap</th>
<th>Time(Secs)</th>
<th>MUMEN</th>
<th>Gap</th>
<th>Time(Secs)</th>
<th>MUMENT</th>
<th>Gap</th>
<th>Time(Secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>320</td>
<td>10.77%</td>
<td>1</td>
<td>4.72%</td>
<td>13</td>
<td>4.72%</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>1.049%</td>
<td>1</td>
<td>3.82%</td>
<td>12</td>
<td>3.82%</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>12.36%</td>
<td>1</td>
<td>4.74%</td>
<td>15</td>
<td>4.74%</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>12.12%</td>
<td>1</td>
<td>2.38%</td>
<td>15</td>
<td>2.38%</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>13.66%</td>
<td>1</td>
<td>6.00%</td>
<td>9</td>
<td>4.06%</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>270</td>
<td>14.70%</td>
<td>1</td>
<td>4.90%</td>
<td>10</td>
<td>4.90%</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>18.38%</td>
<td>2</td>
<td>7.91%</td>
<td>10</td>
<td>4.77%</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>17.17%</td>
<td>1</td>
<td>3.08%</td>
<td>9</td>
<td>3.08%</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>21.51%</td>
<td>2</td>
<td>6.13%</td>
<td>11</td>
<td>3.10%</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>23.84%</td>
<td>2</td>
<td>2.72%</td>
<td>10</td>
<td>2.72%</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>27.61%</td>
<td>1</td>
<td>4.81%</td>
<td>8</td>
<td>1.32%</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>27.02%</td>
<td>1</td>
<td>2.67%</td>
<td>7</td>
<td>2.67%</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>17.96%</td>
<td>1</td>
<td>3.08%</td>
<td>5</td>
<td>3.08%</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>24.61%</td>
<td>1</td>
<td>3.31%</td>
<td>4</td>
<td>3.31%</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>26.14%</td>
<td>1</td>
<td>5.51%</td>
<td>5</td>
<td>4.48%</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>10.77%</td>
<td>2</td>
<td>6.49%</td>
<td>4</td>
<td>5.09%</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>10.49%</td>
<td>2</td>
<td>3.65%</td>
<td>4</td>
<td>3.65%</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-4. Development of hybrid approach (time in seconds).

<table>
<thead>
<tr>
<th># of Railcars</th>
<th># of Units</th>
<th>IMAP</th>
<th>MUMENT-IMAP</th>
<th>MUMENT-IMAP-MUMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Gap</td>
<td>Time</td>
<td>Gap</td>
</tr>
<tr>
<td>80</td>
<td>320</td>
<td>0.00%</td>
<td>97</td>
<td>0.00%</td>
</tr>
<tr>
<td>310</td>
<td>0.00%</td>
<td>158</td>
<td>0.00%</td>
<td>79</td>
</tr>
<tr>
<td>300</td>
<td>0.00%</td>
<td>180</td>
<td>0.00%</td>
<td>192</td>
</tr>
<tr>
<td>290</td>
<td>0.03%</td>
<td>180</td>
<td>0.00%</td>
<td>88</td>
</tr>
<tr>
<td>280</td>
<td>0.00%</td>
<td>180</td>
<td>0.00%</td>
<td>111</td>
</tr>
<tr>
<td>270</td>
<td>1.29%</td>
<td>180</td>
<td>0.00%</td>
<td>76</td>
</tr>
<tr>
<td>260</td>
<td>0.21%</td>
<td>180</td>
<td>0.00%</td>
<td>98</td>
</tr>
<tr>
<td>250</td>
<td>No Sol.</td>
<td>180</td>
<td>1.51%</td>
<td>213</td>
</tr>
<tr>
<td>240</td>
<td>No Sol.</td>
<td>180</td>
<td>0.00%</td>
<td>226</td>
</tr>
<tr>
<td>230</td>
<td>No Sol.</td>
<td>180</td>
<td>1.91%</td>
<td>212</td>
</tr>
<tr>
<td>220</td>
<td>No Sol.</td>
<td>180</td>
<td>1.01%</td>
<td>201</td>
</tr>
<tr>
<td>210</td>
<td>No Sol.</td>
<td>180</td>
<td>1.70%</td>
<td>197</td>
</tr>
<tr>
<td>200</td>
<td>No Sol.</td>
<td>180</td>
<td>1.88%</td>
<td>195</td>
</tr>
<tr>
<td>195</td>
<td>No Sol.</td>
<td>180</td>
<td>2.20%</td>
<td>198</td>
</tr>
<tr>
<td>190</td>
<td>No Sol.</td>
<td>180</td>
<td>4.97%</td>
<td>193</td>
</tr>
<tr>
<td>185</td>
<td>24.36%</td>
<td>180</td>
<td>4.81%</td>
<td>207</td>
</tr>
<tr>
<td>180</td>
<td>59.24%</td>
<td>180</td>
<td>1.89%</td>
<td>199</td>
</tr>
</tbody>
</table>

2.8.2 Results at Different Stages of Hybrid Approach

We then analyzed the gradual improvements in the solution at different stages of the hybrid approach in Table 2-4 and derived the observations below from it. (i) The IPAM takes much less time to find an optimal solution when the number of units is higher than the total capacity of the train. This is because when there are large numbers of units of different types, each railcar can be assigned in the most optimal way, so there will be fewer branches in the branch-and-bound tree. However, as the number of units nears the total capacity of the train, the number of branches will increase in order to find the best fit of the units to the railcars. The quality of the solution generated by IPAM in the time taken by heuristics was variable, and in most cases it was not better than heuristics. It improved with the time as more branches were examined. (ii) The MUMENT search heuristic gave reasonably good solutions for all instances, and when we compared it with the solution of IPAM generated within the same time-interval, the difference became pretty significant. (iii) Intuitively, a good starting point should improve the solution quality as well as the run-time for a branch-and-bound algorithm, but this was not true for all
instances. In one instance, where we found the optimal solution in 97 seconds, the time increased to 101 seconds after providing an initial solution. But in all other cases, it helped in achieving a better solution in the allotted duration. (iv) The second application of the MUMENT search was more effective in the cases where IMAP generated an improved as well as diversified solution. All of these observations indicated that our hybrid approach is the most practical one to use for solving the problem.

2.9 Flexibility of the Model

The IPAM as well as the VLSN search heuristics and hence the hybrid approach discussed in this paper is very flexible and can include other factors that may affect the assignment of units on railcars. In IPAM, additional constraints related to the matching of railcars and units can be included while generating the candidate patterns, while others can be incorporated into the formulation. Heuristics are rather easy to extend since only the arc costs have to be modified for each new cost factor or extra restriction. Although the problem discussed in this paper is static, much less time is taken to run an instance. Therefore, a combination of heuristics and an exact approach can be used to solve the subproblem repetitively in order to optimize the dynamic load planning problem, which we have discussed in Section 2.3. As soon as new information becomes available and planners want to revise the loading plan, they need to update the arc costs in the improvement graph and find negative cycles. In the section below, we provide examples of possible extensions of the problem.

2.9.1 Local Carry Cost

Units are carried by the trucks from the yards to the platform and are loaded with handling machines on the railcars selected by the model. The cost involved in this becomes very significant in several situations, such as with an inefficient handling process, a high-traffic facility, special purpose containers consuming additional resources, etc. This cost easily can be included as an extra term in the IPAM’s objective function. In both of the search heuristics, these costs can be accounted for by modifying
the improvement graph. The costs of arcs signifying the movement of these units should be recalculated when considering these factors.

2.9.2 Handling Time at a Terminal

In our model, we assume that the train under consideration is empty. But as we have discussed in Section 2.3, the current terminal may be the destination for some of the units loaded on the inbound trains and may act as the intermediate terminal for others. This scenario creates the possibility of double handling of outbound units, if the railcars on which these have to be loaded are still occupied with inbound ones. To optimize the LPP with this factor, we need to solve the model every time an event occurs. The event will include the arrival of an outbound unit, the arrival of a pick-up truck, etc. The objective function should be modified to include the probability distribution of all the events. For this scenario, the MUMEN search algorithm will work better, as once an optimal solution is found, the number of exchanges will be very few with the occurrence of new events.

2.9.3 Daily or Weekly Load Plan

In this paper, we have solved the LPP for a single train. Sometimes, partial information about future trains also is known and the plan for an entire week for multiple trains is needed. This scenario can be incorporated into the IPAM by adding an extra dimension (train) into the variables and by modifying the objective function to include the cost of all the trains into the planning horizon. Due to the element of uncertainty attached to a future train, the model output may be suboptimal. To reduce the possibility and extent of suboptimality, we should assign a weight distribution to each train’s cost. In the MUMENT searches, we can extend the underlying improvement graph, and the new arc’s cost should be such that it favors the flow toward the train that is earliest in the planning period.
2.10 Conclusions

In this paper, we solve the load planning problem, which is considered very important in the industry, as a little improvement in the assignment will result in significant savings. Considerations of all the issues of real-life applications make the problem very complex and non-linear in nature. We use a unique MIP model which captures all relevant issues and gives good solutions for some of the instances, but the quality deteriorates as the demand and capacity get within same range. This motivates us to develop the VLSN heuristics: SUMEN and MUMEN searches. The MUMEN search is effective in all instances, but the scattered feasible region forces it to a local optimum. To overcome this problem, we extend the algorithm using the tabu search, which improves the solution quality and makes it more robust. We finally suggest a hybrid algorithm that combines the best characteristics of the MIP as well as of the VLSN search. It improves the run-time and solution quality even further. This hybrid approach is applicable for the static version as well as for the dynamic version of the problem, which requires several revisions of the plan. To summarize, our hybrid approach is the most effective solution for real-life problems and easily can be extended to cover additional requirements.
CHAPTER 3
SUBSET DISJOINT MINIMUM COST CYCLE DETECTION

3.1 Introduction

Minimum cycle or negative cycle problems play a central role within the field of routing and scheduling problems. They arise frequently in a wide variety of applications either as stand-alone problem or as a part in more complex problem settings and thereby attracting the attention of practitioners as well as researchers. We will see some applications of the problem later in this chapter. The subset disjoint minimum cycle problems typically constitute from a network represented by a directed graph \( G = (N, A) \), where \( N \) is the set of \( n \) nodes divided into subsets \( S = S_1, S_2, \ldots, S_k \), and \( A \) denotes the set of \( m \) arcs between nodes. Each arc \((i, j) \in A\) has an associated cost \( c_{ij} \), also called length or weight of the arc. A cycle \( q = \{i_1, i_2, \ldots, i_k, i_1\} \) is a cycle of length \( k \) in the graph \( G \), where \( i_j \in N \), and \((i_j, i_{j+1}) \in A, \forall j = 1, k - 1 \). The cost of the cycle \( q \) is sum of cost of all arcs present in the cycle as given below: 

\[
c(q) = \sum_{j=1}^{k-1} c_{i_j, i_{j+1}} + c_{i_k, i_1}
\]

A cycle, \( q = \{i_1, i_2, \ldots, i_k, i_1\} \) in \( G \), is called simple cycle if each node has only one incoming arc and one outgoing arcs among all arcs of the cycle, i.e. \( i_j \neq i_k \forall j, k \in \{1, 2, \ldots, k\} \). In this chapter, we are concerned with finding the subset-disjoint cycle in a graph \( G \), with nodes divided among subsets \( S = \{S_1, S_2, \ldots, S_k\} \), with \( S_1 \cup S_2 \cup \ldots \cup S_k = N \), and \( S_j \cap S_k = \phi, \forall j, k \in \{1, 2, \ldots, k\} \). A cycle \( q \) is called subset disjoint cycle if it is simple as well as at most one node of any subset is present in the cycle. In other words, 

\[
|N(q) \cup S_k| \leq 1 \forall k \in \{1, 2, \ldots, k\},
\]

where \( N(q) \) is the set of nodes involved in the cycle. The subset disjoint minimum cost cycle (SDMCC) problem is to find the subset disjoint cycle \( q \) in \( G \) with minimum cost \( c(q) \). The subset disjoint negative cost cycle (SDNCC) problem is a special case of the minimum cycle problem where total cost of the cycle must be negative, i.e. \( c(q) < 0 \). The examples of SDMCC is given in Figure 3-1. The graph in Figure 3-1 has node 1 in subset 1, node 2 in subset 2, node 3 in subset 3, and nodes 4 and 5 in subset 4. The SDMCC, which is the optimal solution of SDMCC...
problem in the given network, is the cycle (1-2-3-1) of cost 10. The overall minimum cycle of the graph is (1-2-4-5-3-1) of cost 2, but we eliminate it from the feasible set as it is not subset disjoint (nodes 4, and 5 belong to the same subset). In this example, there doesn’t exist any negative cycle.

![Figure 3-1. Illustrating subset-disjoint minimum cost cycle.](image)

We illustrate the subset disjoint negative cost cycle in Figure 3-2. We keep the node partitions same as in Figure 3-1, and change only the cost of arcs to simplify the illustration. In this network, we have two feasible negative cycles: (i) (1-2-3-1) of cost -4, and (ii) (1-2-5-3-1) of cost -10. Therefore, the optimal cycle of SDNCC problem is (1-2-3-5-1). There exists one more negative cycle (1-2-4-5-3-1) of cost -12, but we eliminate it from consideration as it not subset-disjoint. It should be noted that in this example the minimum is same as the negative cycle (1-2-3-5-1).

From the given definitions and illustrations, we can easily conclude following two relations between SDMCC problem and SDNCC problem:

1. The feasible set of negative cycle problem will be a subset of the feasible set of minimum cycle problem in the same graph G.
2. If there exists a feasible solution for the negative cycle problem, the optimal solution of minimum cycle and negative cycle will be same.
The subset-disjoint minimum and negative cost cycle problems are usually the core structure of most of the routing based optimization problems, and multiple-exchange based neighborhood search algorithms. It has been proved NP-complete in [33]. We next discuss some of its direct and indirect applications.

1. **Prize collecting travelling salesman problem** [8] A salesman travels between pairs of cities at a cost depending only on the pair, and gets a prize in every city that he visits. However, he needs to pay a positive penalty for every city that he fails to visit. The objective of the problem is to maximize his earning (prize collected - travel costs - penalties). There is no constraint on the minimum reward. We can reduce this problem to the SDMCC problem by considering each city as node. We also assume that each node has its own subset, i.e. each subset contains only one node. Arcs \((i, j) \in A\) are constructed between node pair \((i, j)\), if the salesman can travel to city \(j\) after city \(i\). The cost of each arc \((i, j) \in A\), is assigned as the sum of three cost components: (i) travel cost from city \(i\) to city \(j\), (ii) negative of the penalty of not visiting city \(i\), and (iii) prize collected for visiting city \(j\). After the construction of the network, we apply the SDMCC algorithms to find the minimum cycle which has one to one correspondence to the optimal solution of the prize collecting travelling salesman problem.

2. **Very large scale neighborhood exchange search** [6] The very large scale neighborhood search is generally used in the context of solving partitioning problems, which are highly combinatorial and huge size problems. Given a set \(S\) of \(n\) elements, the objective to find a partition \(T = \{T_1, T_2, ..., T_k\}\) of \(S\), where \(T\) is set of subsets of \(S\). Each pair of subsets must be mutually exclusive, and the union
of all subsets must be completely exhaustive. Each subset of the partition is also associated with some costs \( c(T_k) \), where \( T_1 \cup T_2 \cup \ldots \cup T_k = S \). The cost of the subset is a function of all the elements inside the subset \( T_k \). So, we aim to find the partition of \( S \), such that the total sum of cost all subsets is minimized.

One of the most effective heuristics to solve the partition problem is very large-scale neighborhood search (VLSN). In these neighborhood search algorithms, first a feasible solution is constructed, and then, a series of exchanges of elements among subsets is found such that changing the subsets of elements as per those exchanges improves the total cost of the partition. In elementary search algorithms only 2-exchanges are determined which is also called swapping. However, more advanced search algorithms like VLSN search, performs multiple exchanges at the same time. The process of finding the best set of exchanges of arbitrary length can be modeled as the subset disjoint negative cycle problem. In VLSN search, it is done by constructing an improvement graph based upon the incumbent feasible solution. In the improvement graph, nodes and arcs are constructed along with the cost assignment on the arcs. The cost of the arcs between the nodes of the same subset is initialized to infinity which forces the cycle to be subset-disjoint. Cost of other arcs in the improvement graph is defined such that the total costs of all exchanges becomes equivalent to the change in the total costs of partition if elements are redistributed among subsets accordingly. The best set of exchanges is found by determining a subset disjoint minimum cycle in the improvement graph. The improvement graph is constructed in such a way that path exchanges can also be represented as cycle exchanges. The complete VLSN search algorithm has been described in Chapter 2.

Although SDMCC or SDNCC problems are very commonly used directly or indirectly in several algorithms, we don’t have any literature dedicated to them. Generally, some dynamic programming is used which is customized to particular problems and may not be applicable to other class of problems. In this chapter, we have developed several generic algorithms which can be applied in all problems where minimum cycle is to be found repetitively. The contributions of this paper include:

1. We suggest four exact algorithms for the SDMCC problems and SDNCC problems. These algorithms can be seen as the generalized version of the dynamic programming algorithms generally used for the constrained shortest path problems.

2. Network reduction: The complexity of exact algorithms depends on the size of network (nodes and arcs). We introduce several strategies to reduce the size of the network and thereby, to decrease the complexity of the algorithm.
3. **Algorithm acceleration:** We present some new and some existing strategies used in other applications, to improve the practical running time of the algorithms. These strategies don’t improve the worst case complexities of algorithms; however, they prove to be very effective in practical applications.

4. The exact algorithms may perform very efficiently even on large instances, the solution time must be even lower than fraction of seconds in some algorithms such as VLSN search. We propose several heuristics based on the exact algorithms by controlling different parameters of the exact algorithms.

5. We finally give an extensive computational analysis by running all algorithms on different classes of problems.

### 3.2 Network Reduction

In this section, we describe some approaches to decrease the size of the network. These approaches are applicable to all algorithms discussed in this chapter. These approaches are based on simple observations, but they significantly reduce the size of network. We use them as the preprocessing step of all the algorithms. These consist of removing arcs that cannot be a part of optimal cycle. We next discuss all these properties along with examples. We state that an arc \((i, j) \in A\) should be deleted from the network if any of the following properties are satisfied:

1. Subset of node \(i\) = Subset of node \(j\).
2. Node \(i\) has no incoming arc.
3. Node \(j\) has no outgoing arc.
4. For all \(k \in V\) such that \((j, k) \in A \land (i = k \lor s(k) \neq s(i)) \land s(k) \neq s(j)\) there exists a \(q \in S_{s(j)}, q \neq j\) such that \((q, k) \in A, (i, q) \in A\) and \(c_{iq} + c_{qk} < c_{ij} + c_{jk}\).
5. For all \(k \in V\) such that \((k, i) \in A \land s(k) \neq s(i) \land (j = k \lor s(k) \neq s(j)\) there exists a \(q \in S_{s(i)}, q \neq i\) such that \((k, q) \in A, (q, j) \in A\) and \(c_{kq} + c_{qj} < c_{ki} + c_{ij}\).

**Proof:** The first three properties are obvious. To prove that the property 4 is sufficient, assume that it is satisfied and that we have a subset disjoint cycle \(q\) that uses the arc \((i, j)\). Let \((j, p)\) be the arc after \((i, j)\) in the cycle. Then we can obtain a cycle \(q'\) such that \(c(q') < c(q)\) by exchanging arcs \((i, j)\) and \((j, p)\) with arcs \((i, q)\) and \((q, p)\), where \(q\) is a node in \(S_{s(j)}\) such that \(c_{iq} + c_{qp} < c_{ij} + c_{jp}\) (such a node exists according
to 4). Therefore the cycle $q$ that we started out with could not be optimal. The proof of condition 5 follows the same pattern.

Notice that we can exchange the less-than signs with less-than-or-equal signs in fourth and fifth properties. Then, we can no longer say that $(i,j)$ cannot be part of an optimal solution, but if $G'$ is the graph obtained by removing $(i,j)$ from $G$ we can say the optimal solution to SD(M/N)CC problems in $G'$ also an optimal solution to SD(M/N)CC problems in $G$.

The preprocessing algorithm is implemented in all algorithms discussed in this chapter in the function $reduction$. It checks the arcs in the graph one by one and removes the one that satisfies any of these five properties. It can be worthwhile to run several passes of the algorithm (a pass being a check of all arcs). If arcs has been removed in the previous pass it may be possible to remove even more arcs in the next pass. If the graph is very dense we keep running the preprocessing algorithm till the reduction in the network size falls below a certain limit. We next discuss the exact algorithms to find the SDMCC or SDNCC.

### 3.3 Exact Algorithms for Subset Disjoint Minimum Cost Cycles

We describe four exact algorithms for each version of the problems; SDMCC, and SDNCC. All these algorithms are based on dynamic programming approach. All these algorithms are based on a labeling scheme, which is discussed next in this section. First two algorithms can be viewed as the generalized version of the labeling algorithms for the constrained shortest path problems. In the first algorithm, Dynamic Labeling Algorithm, we dynamically add new labels and delete old ones, finally converging to the optimal set of labels. In the second algorithm, Fixed Labeling Algorithm, we store only those labels which may come in the final solutions. Third and fourth algorithms can be viewed as variants of all-pair shortest path algorithms, and therefore we call them "All-pair like Pull Algorithms", and "All-pair like Push Algorithms" respectively. We
discuss the details of each algorithm sequentially in the next few sections starting with
the common labeling scheme.

3.3.1 Labeling Scheme

All the algorithms discussed in this chapter are based on the common labeling
scheme. In these algorithms, we find some paths from source to all nodes, and convert
them to cycles by regularly checking some backward edges to the source. For any node
i, and source s, a path \( p_i \) is a sequence of nodes \( s = i_0, i_1, \ldots, i_l = i \), which ends at the
node i. We next define some terms which will be regularly used in our algorithms.

\( l \): Label - Each label indicates a partial path starting from the source to a node. It is a
set of quadruple \( (\eta, c, R, p) \).

\( \eta(l) \): End node of a label \( l \) - It indicates the last node of the partial path indicated by the
label \( l \).

\( c(l) \): Cost of a label \( l \) - It indicates the total cost of the partial path indicated by the label \( l \).

\( R(l) \): Resource vector of label \( l \) - It is a binary vector with one element for each subset.
1 indicates that the partial path indicated by the label \( l \) has covered that subset, while 0
means it has not. It is used to determine if the current partial path should be extended to
a particular subset.

\( p(l) \): Parent of label \( l \) - It denotes the parent label of the label \( l \). It is used to find the
actual path once the final solution is found. If a label doesn’t have any parent, we put -1
for as its parent.

\( U \): Set of untreated labels - It is the set of all labels, which are yet to be examined to
determine if the path represented by them should be extended, or if these labels should
be deleted, or if these are the best labels and will represent the best cycle by joining the
arc from the end node of the label to the current source.

\( L_i \): set of labels ending at node \( i \) - It represents the set of all labels which have \( i \) as the
last node represented by their partial path. It should be noted that there can be multiple
labels ending at the same node. However, they will vary either in the resource vector or in the cost.

$\delta^+(i)$: It indicates the set of outgoing arcs from the node $i$.

$\delta^-(i)$: It indicates the set of incoming arcs to the node $i$.

$s(i)$: It indicates the subset to which node $i$ belongs.

**Dominance of labels**

Let $l_1$, and $l_2$ be two labels corresponding to two different paths from the source $s$ to a node $i$, i.e. $l_1, l_2 \in L_i$. Then, the label $l_1$ dominates label $l_2$ if and only if $c(l_1) \leq c(l_2), R(l_1)_k \leq R(l_2)_k, \forall k = 1, 2, \ldots, K. R(l_1)_k$ denotes the $k^{th}$ element of the resource vector. In other words, a path from source to node $i$ will dominate another if and only if it has lower cost and visits a subset of nodes visited by the second path. If the costs of two paths are same, we give the preference to the one which visits fewer numbers of nodes.

**Efficient labels**

A label $l_1$ corresponding to a path from source to node $i$ is efficient if it is not dominated by any other label from the set $L_i$. The path corresponding to the efficient label is called efficient path.

**Label ordering in the set of unprocessed labels**

The labels in the set of unprocessed labels, $U$, are sorted in non-decreasing order using the ordering scheme $<_L$ defined as follows: $l_1 <_L l_2$ if and only if $(R(l_1) <_R R(l_2))$ OR $(R(l_1) = R(l_2) AND c(l_1) < c(l_2))$. The comparison $<_R$ is the lexicographic comparison of bit vectors represented by the resource vectors of both labels.

We use the above mentioned labeling concepts in all our algorithms. Now, we discuss each algorithm for SDMCC as well as SDNCC problems. We also discuss some algorithm acceleration techniques to improve the practical running time of these algorithms.
3.3.2 Dynamic Labeling Algorithm (DLA) for SDMCC Problems

In this section, we propose a generalized algorithm based on the concepts of dynamic programming algorithms for constrained shortest path problems. Our algorithms find the subset-disjoint paths, which are elementary paths in the graph which uses at most one node from any subset. The definition is very similar to that of subset-disjoint cycles, and any subset-disjoint path from the source to node $i$, can be converted to corresponding cycle just by including the arc $(s, i)$. We assume that the arc exists between each pair of node. If it doesn’t, we create an arc with a very high cost, eliminating its possibility of occurring in the final solution. The set of all these paths make a many-to-one correspondence with the corresponding set of subset disjoint cycles. It should also be noted that for every SDMCC $q$, the path induced $p$ by eliminating any arc $(i, j) \in A(q)$, is a subset disjoint minimum cost path from the node $j$ to node $i$ for all pairs of $(i, j) \in N$. In other words, if we can find all subset disjoint minimum cost paths, we can easily convert them to cycle and select the one which has minimum cost. The selected cycle must be an optimal solution to the SDMCC problem. Our algorithm is based on the same concept. The details of the algorithm are given in Algorithm 1.

We have used several terms and subroutines in the algorithm described above to keep it clean. We describe the functions of these subroutines separately. minCostCycle: It stores the cost of the best minimum cost cycle found at any stage. bestLabel: It stores the label corresponding to the best minimum cost cycle found at any stage. first($U$): It returns the first element or label from the set of unprocessed labels $U$ sorted according to the ordering described in the section 3.3.1, and removes it from the list. not-dominated($l'$, $L_{\eta(r)}$): It returns true if the label $l'$ is not dominated by any of the labels in the set ($L_{\eta(r)}$). remove_dominated($l'$, $L_{\eta(r)}$, $U$): If the function $l'$ is not dominated by any label present in the set of labels with the same end node, it is possible


**Algorithm 1 SDMCCA-DLA**

1: reduction();
2: minCycleCost = \( \infty \)
3: bestLabel = NULL
4: **for all** \( \alpha \in N \) **do**
5: \( U = \{ (\alpha, 0, R, -1) \} \);
6: \( L_\alpha = \{ (\alpha, 0, R, -1) \} \);
7: \( L_i = \emptyset \quad \forall i \in N, i \neq \alpha \)
8: **while** \( U \neq \emptyset \) **do**
9: \( l = \text{first}(U) \);
10: **for all** \((i, j) \in \delta^+(\eta(l))\) **do**
11:    **if** \( j = \alpha \) **then**
12:       **if** minCycleCost > \( c(l) + c_{i,j} \) **then**
13:          minCycleCost = \( c(l) + c_{i,j} \);
14:          bestLabel = \( l \);
15:       **end if**
16:    **end if**
17:    **if** \( R(l)_j = 0 \) **then**
18:       \( l' = (j, c(l) + c_{i,j}, R(l) + e_j, l) \);
19:       **if** not_dominated\( (l', L_\eta(l')) \) **then**
20:          remove_dominated\( (l', L_\eta(l'), U) \);
21:       \( U = U \cup \{ l' \} \);
22:       \( L_\eta(l') = L_\eta(l') \cup \{ l' \} \);
23:    **end if**
24: **end for**
25: **end while**
26: **end for**

that there is some probability that \( l' \) is dominating other labels. This function removes all those labels from the set \( L_\eta(l') \), and \( U \) which are dominated by \( l' \).

In each iteration of the DLA a node \( \alpha \in N \) is considered as source, and the best cycle (minimum cost cycle) which must include it is found. Starting from the source, it first creates a label with the source as the end node, 0 cost, resource vector with 1 at the position of subset of the source node, and -1 (NULL) as the parent. The newly created label is inserted into the set of unprocessed labels as well as in the set of labels ending at the source node. The set of labels at all other nodes are initialized to empty set. Thereafter, each label from the set of unprocessed label is examined or processed till it becomes empty. While examining any label \( l \in U \), all outgoing arcs \((i,j) \in O(\eta(l))\) are
traversed and following conditions are checked to verify if the label $l$ can be extended in the direction of the arc $(i, j)$: (i) if the resource vector of label $l$ already contains the subset to which node $j$ belongs, and (ii) if the potential label at the node $j$ is dominated by labels already present at that node, i.e. labels of the set $L_j$. If both conditions are false, then new label at node $j$ is created and inserted in the set $U$ as well as $L_j$. As there is some probability that, the newly inserted node may be dominating labels already existing in these sets, we examine these sets and delete all dominated labels. This process continues till all labels get exhausted and the set $U$ becomes empty. At the start the cost of minimum cycle is initialized to a very high value, and the best cycle to a null value. We verify the existence of better cycles whenever we start processing a new label. We check if the sum of the cost at the current label $l$ and the cost of the arc $(\eta(l), \alpha)$ is lower than the current best cycle value. If the result is true, the best cost as well as the best cycle is updated.

The correctness of the algorithm is clear from the construction of the algorithm as we check the minimum cost cycle from each node considering as a source at a time. While finding the SDMCC, we find all subset disjoint minimum path from the source to all other nodes, and convert them to the cycle. The algorithm will definitely terminate as all labels are selected for processing in a specific order as described in the Section 3.3.1, which ensures that strictly 'larger' labels are created from the current label. In other words a label already processed will never be created again, and hence, no cycling will occur, which proves the termination of the algorithm. The complexity of the dynamic label algorithm is $O(|N||A|2^K)$, which is clearly exponential in $K$. Although, even with the better data structure the worst case complexity cannot be improved, but we can definitely improve the running time by accelerating some algorithmic steps as described in the next section.
3.3.3 Algorithm Acceleration for DLA

As described earlier, in each iteration of the DLA, one node is selected as the source and the best minimum cost cycle is found which must involve this node. Suppose we consider $i_1$ as the current source node, and find the optimal cycle $q = (i_1 = s, i_2, \ldots, i_l)$. It is clear that the same optimal cycle will also be found whenever any node $i_j \forall j = 1, 2, \ldots, l$ of the cycle is considered as the source node. Hence, the same cycle is found at least $l$ times. Using this observation, we can eliminate this redundancy without eliminating any feasible solution. Suppose all nodes are numbered sequentially from 1 to $n$. When we consider the node $i \leq n$ as the source node, we consider the arcs incident to only nodes $j \geq i$. By this strategy, we eliminate all those cycles which include any node $k \leq i$, which have already have been found while considering the node $k$. The current algorithm can be modified to incorporate this strategy by dynamically changing the graph for each node considered as a source. Suppose $G^s = (N^s, A^s)$ denote the graph induced by nodes $s, s + 1, \ldots, n$. In other words, $N^s = s, s + 1, \ldots, n$, and $A^s = (i, j) \in A : i, j \geq s$. Hence, we just need to replace $N$ by $N^s$, and $A$ by $A^s$ in the algorithm. However, changing the graph dynamically may be time-expensive, so we just replace the line 10 of the the algorithm by following:

for all $(i, j) \in \delta^+(\eta(l))$ and $j \geq \alpha$ do

3.3.4 DLA with Acceleration for SDNCC Problems

The dynamic labeling algorithm for the SDNCC problem is almost same as that for the SDMCC problems. However, the algorithm can be accelerated at several levels by using the property specific only to negative cycles.

Property 1: For any negative cost cycle $q = (i_1, i_2, \ldots, i_l, i_1)$ of length $l$ in $G$, there exists $j \in 1, 2, \ldots, l$ such that $c(p_k) < 0 \forall k = 1, 2, l$, where $p_k = (i_j, i_{j+1}, \ldots, i_{j+k})$, and all subscripts arithmetic is performed modulo $l$.

Using the above property, we can conclude that for any optimal solution of SDNCC problem, there must exist a node $i$ such that every path in the subgraph of $G$ obtained
by only the nodes and arcs present in the cycle, has total negative cost. We can implement this property in the DLA by restricting the assignment of sources to only those nodes which has at least one outgoing negative arcs. Moreover, since the cost of optimal path always remains negative, we will also restrict the extension of current labels through only those arcs which keeps the total cost negative.

We have two acceleration techniques which can be used for SDNCC problems: (i) as described in Section 3.3.3, and (ii) described using property 1. Unfortunately, we can not use both techniques at the same time. In first property we assume that all cycles with lower numbered nodes must have been found when lower numbered nodes was considered as source. While, if we use the second technique, some cycle may start from the larger node and pass through a smaller node, which doesn’t have any outgoing arc with negative cost, to complete a negative cost cycle.

Generally, a network contains very small percentage of negative cost arcs, so if we use the second acceleration technique, we will eliminate many source nodes and reduce the number of labels created while label extension process. However, if the percentage of negative arcs is very high, we should use the first acceleration technique as negative cost criteria would be satisfied by almost all nodes. The worst-case complexity of DLA for SDMCC and SDNCC problems remain same. We next give the details of the algorithm for completeness.

3.3.5 Fixed Labeling Algorithm (FLA) for SDMCC Problems

In this section, we propose another variation of the generalized algorithm based on the concepts of dynamic programming algorithms for constrained shortest path problems. As DLA, it also finds the subset-disjoint paths between two nodes and converts them to cycle using the backward arc. It uses the same functions described in Section 3.3.2 for the DLA, however, uses different strategy to store the labels. In the fixed label algorithm, like DLA, we maintain a set of unprocessed labels \( U \), but unlike DLA, we maintain the set \( L_i \) of non-dominated, and processed labels at every node.
Algorithm 2 SDNCCA-DLA

1: reduction();
2: minCycleCost = \infty
3: bestLabel = NULL
4: for all \(\alpha \in N\) with \(c_{\alpha i} < 0\) for some \(i \in N\) do
5:   \(U = \{(\alpha, 0, R, -1)\};\)
6:   \(L_\alpha = \{(\alpha, 0, R, -1)\};\)
7:   \(L_i = \emptyset \ \forall \ i \in N, i \neq \alpha\)
8: while \(U \neq \emptyset\) do
9:   \(l = \text{first}(U);\)
10:   for all \((i, j) \in \delta^+(\eta(l))\) AND \(c(l) + c_{ij} < 0\) do
11:     if \(j = \alpha\) then
12:       if \(\text{minCycleCost} > c(l) + c_{ij}\) then
13:         \(\text{minCycleCost} = c(l) + c_{ij};\)
14:         \(\text{bestLabel} = l;\)
15:     end if
16:   end if
17:   if \(R(l_j) = 0\) then
18:     \(l' = (j, c(l) + c_{ij}, R(l) + c_j, l);\)
19:     if \(\text{not_dominated}(l', L_{\eta(l')})\) then
20:       \(\text{remove_dominated}(l', L_{\eta(l')}, U);\)
21:       \(U = U \cup \{l'\}; L_{\eta(l')} = L_{\eta(l')} \cup \{l'\};\)
22:     end if
23:   end if
24: end for
25: end while
26: end for

\(i \in N\). In DLA \(L_i\) stores processed as well as unprocessed node without considering the domination criteria. So, the main difference between FLA and DLA is the way the set \(L_i\) is maintained. In DLA, we do not guarantee that the labels in these sets are non-dominated and we remove labels from the sets as we discover new labels that are dominating existing labels in \(L_i\). In FLA, we guarantee that the labels in \(L_i\) are non-dominated and we never have to remove any label from the sets once they are entered. We know that labels in \(L_i\) are non-dominated because of the order in which we treat labels. The ordering policy and dominance criteria make sure that if a label \(l_1\) is ordered before label \(l_2\), i.e. \(l_1 <_L l_2\), then label \(l_2\) cannot dominate label \(l_1\). The
The acceleration technique described in Section 3.3.3 is valid for FLA, too. We include them in its algorithmic framework.

The corresponding algorithm for SDNCC problems can be obtained by making same changes as we did while converting the DLA of SDMCC problem to SDNCC problem. More explicitly, we replace the line 3 by \textbf{for all } \alpha \in N \text{ with } c_{\alpha i} < 0; i \in N \text{ do}; and line 10 by \textbf{for all } (i,j) \in \delta^+(\eta(l)) \text{ AND } c(l) + c_{ij} < 0 \text{ do}.

The complexity of DLA and FLA for both versions of the problem remains same. However, there are two main differences between these two algorithms: (i) number of labels stored in DLA approach is much higher than that of FLA, and (ii) the number of checks performed for establishing the dominance of labels is comparatively higher in FLA. We discuss the effect of these differences in the computational analysis in Section 3.5.

3.3.6 All-Pairs like Pull Algorithms (APPull)

In this section, we suggest a different approach to solve the SDMCC problem. but the basic definitions of all functions remain consistent. Similar to the first two algorithms, in this algorithm too, each node is considered once as a source, and one-to-many subset-disjoint shortest paths are found. These paths are then converted to cycles and the minimum cost cycle is chosen as the optimal cycle. However, in this algorithm we choose a different ordering scheme to order the labels present in the set of unprocessed labels. In DLA and FLA, the bit wise comparison was done between the resource vectors of two labels to decide the ordering. Here, we use the length of path represented by the label as the ordering criteria, where the length of a path is defined as the number of subsets visited by the path. It can also be viewed as the number of ones in the resource vector of the label. So, \( l_1 <_L l_2 \) if \( \sum_{k=1}^{K} R(l_1)^k < \sum_{k=1}^{K} R(l_2)^k \).

In each step of the algorithm, a label with least path length is chosen and extended to the another label if all conditions are satisfied. It is clear that the newly created label has the length exactly one more than the parent label. This label selection strategy
Algorithm 3 SDMCCA-FLA

1: reduction();
2: minCycleCost = ∞
3: bestLabel = NULL
4: for all α ∈ N do
5:   U = {(α, 0, R, −1)};
6:   L_i = ∅ ∀i ∈ N
7: while U ≠ ∅ do
8:   l = first(U);
9:   if not dominated(l, L_η(l)) then
10:      L_η(l) = L_η(l) ∪ {l};
11:      for all (i, j) ∈ δ^+(η(l)) and j ≥ α do
12:         if j = α then
13:            if minCycleCost > c(l) + c_{i,j} then
14:               minCycleCost = c(l) + c_{i,j};
15:               bestLabel = l;
16:         end if
17:      end if
18:      if R(l)_j = 0 then
19:         l' = (j, c(l) + c_{i,j}, R(l) + e_j, l);
20:         if not dominated(l', L_η(l')) then
21:            U = U ∪ {l'};
22:         end if
23:      end if
24:   end for
25: end if
26: end while
27: end for

along with the label ordering, ensures the termination of the algorithm in finite time as a path can be of maximum length n.

We utilize the new ordering strategy very efficiently in the "all-pair" like pull algorithm, which can be viewed as a generalization of dynamic programming algorithms for "all pair shortest path problems" discussed in [5]. For each source node i in N, we first find all paths of length 1, then extend them to path of length 2, and so on till we reach the maximum path length n. It is not a true generalization as in our algorithms, we generate all paths of length k only from the current source i ∈ N and move to finding paths of length k + 1. However, in actual "all-pair shortest path dynamic algorithms" all
paths of length \( k \) are found before moving to finding paths of length \( k + 1 \). The reason for this variation is the very high level of memory requirement in the true generalized version of the algorithm, which renders it ineffective in practice.

The APPull algorithms use the same label dominance criteria and has the same worst case complexity. It uses the fact that since, there are at most \( K \) subsets in the network, there can be at most \( K \) nodes and \( K \) arcs in any subset disjoint cycle. So, we need to consider only subset-disjoint paths upto length \( K \). The algorithm calculates minimum cost subset disjoint paths between each pair of nodes by first calculating subset-disjoint paths of length at most \( k \), \( \forall k = 2, 3, ..., K - 1 \) sequentially. After calculating subset-disjoint paths of length at most \( k \) between node pairs \( i, \text{and} j \in N \), we extend it to paths of length \( k + 1 \) by examining each arc \((r, j) \in A\), where \( r \neq i \). It is clear that if there is an subset disjoint efficient path of length \( k + 1 \) from \( i \) to \( j \) using arc \((r, j)\), then its subpath from \( i \) to \( r \) is also a subset disjoint path of length \( k \). So, using the induction we can conclude that the algorithm generates all possible efficient paths between each pair of nodes.

We now discuss the "all-pair like pull" algorithm for SDMCC problems. The details are given in Algorithm 4. Like the DLA and FLA, we initialize the best cycle cost with infinity and best cycle to a null value. We assign each node \( \alpha \in N \) as source once, and find the cycles which must contain it. In line 5-6, we create a label at each node \( i \) which has an incoming arc from the source \( \alpha \). The newly created label, \( l \) has \( i \) as the end node, cost of the arc \((\alpha, i)\) as the cost of the label, the resource vector with \( R(l)(\alpha) = 1 \) and \( R(l)(i) = 1 \), and -1 as the parent label. In line 7-12, we check the cost of all cycles of length 2, and update the best label and best cost, if required. In line 14-30, all these labels (representing paths of length 1) are extended using the outgoing arcs of the corresponding end node. In this algorithm, the set of labels \( L_i^{k-1} \) stores labels with end node \( i \) with corresponding path length \( k' \leq k - 1 \). We choose a node \( j \in N \), and examine all nodes \( i \in N \) which has an incoming arcs \((i, j) \in A \). If
there exists some labels at node $i$ with in the label set $L_i^{k-1}$, we check if the current label can be extended to node $j$ using the arc $(i, j)$. As earlier while checking if extension is possible, we examine the resource vector, and the cost of the label. If all conditions are satisfied, we perform the dominance test, where we check if the newly created label at node $j$ is dominated by already existing labels at node $j$. The comparison of the newly created label with corresponding path length $k$ is done with all labels present at the node $j$ with path length $k' \leq k$. If it is not being dominated, we insert it into the label set at node $j$ with path length $k$, i.e. in the set $L^k_j$ and delete all labels dominated by the new label. Once, the subset disjoint minimum cost paths are calculated upto length $K$, best labels at each node $i \in N$ are stored in the set $L^k_i$ for the path of length $k$. In line 31-38, we compare the costs of cycles formed by including the arc $(i, \alpha)$ in all subset disjoint paths at corresponding to the best labels at each node $i$. It should be noted that the algorithmic framework also includes the acceleration technique presented in Section 3.3.3.

The APPull algorithms can be easily modified to solve the SDNCC version of the problem. To differentiate these algorithms we call them, APPull-M and APPull-N algorithms, respectively. The APPull-N algorithms use the acceleration strategy specific for negative cycles assuming that the percentage of negative cost arcs in the network is very low. The extend-possible function in line 13, examines if the extended label’s cost remains negative, besides checking the resource vector. We present the details of the algorithm for completeness of the discussion.

The APPull algorithms can be converted to a pure generalized version of all pair shortest path algorithms, by exchanging the "for loop" of line 3 and line 14. However, pure version stores very large number of labels, and slows down the complete algorithm. The APPull algorithm reduces the memory requirement by storing fewer labels, and generates solutions in much lower running time. We show the detailed computational results in Section 3.5.
Algorithm 4 All-pairs-like-pull (APPull) algorithm for SDMCC problems

1: reduction();
2: minCycleCost = ∞
3: bestLabel = NULL
4: for all $\alpha \in N$ do
5:   $L^k_i = \emptyset \quad \forall i \in N, k \in 1, ..., m$;
6: for all $i \in N$, $(\alpha, i) \in A, \text{subset}(\alpha) \neq \text{subset}(i)$ do
7:   $L^1_i = l = \{(i, c_{\alpha i}, e_{\text{es}(\alpha)} + e_{\text{es}(i)}, -1)\};$
8:   if $(i, \alpha) \in A$ then
9:     if minCycleCost > $c(l) + c_{i,\alpha}$ then
10:        minCycleCost = $c(l) + c_{i,\alpha}$;  
11:        bestLabel = $l$;
12:     end if
13:   end if
14: end for
15: for $k = 2$ to $K$ do
16:   for all $j \in N$ and $j > \alpha$ do
17:     for all $i \in N$ and $i > \alpha$ do
18:       if $(i, j) \in A$ then
19:         for all $l \in L^{k-1}_i$ do
20:             if extend_possible($l, (i, j)$) then
21:               $l' = \text{extend}(l, (i, j))$;
22:               if not_dominated($l', \bigcup_{k'=1}^{k} L^{k'}_j$) then
23:                 remove_dominated($l', L^{k'}_j$);
24:               $L^k_j = L^{k'}_j \cup \{l'\};$
25:             end if
26:         end if
27:       end if
28:     end for
29:   end if
30: end for
31: end for
32: for all $(i, \alpha \in A)$ and $i > \alpha$ do
33:   for all $l \in L^{K-1}_i$ do
34:     if minCycleCost > $c(l) + c_{i,\alpha}$ then
35:       minCycleCost = $c(l) + c_{i,\alpha}$;  
36:       bestLabel = $l$;
37:     end if
38:   end for
39: end for
40: end for
Algorithm 5  All-pairs-like-pull (APPull) algorithms for SDNCC problems

1: reduction();
2: minCycleCost = ∞
3: bestLabel = NULL
4: for all α ∈ N do
5:     L_\alpha^k = {} ∀ i ∈ N, k ∈ 1, ..., K;
6:     for all i ∈ N, (α, i) ∈ A, subset(α) ≠ subset(i), c_{ai} < 0 do
7:         \( L_1^i = \{(i, c_{ai}, e_{s(α)} + e_{s(i)}, -1)\} \);
8:         if (i, α) ∈ A then
9:             if minCycleCost > c(l) + c_{i,α} then
10:                minCycleCost = c(l) + c_{i,α};
11:                bestLabel = l;
12:         end if
13:     end if
14: end for
15: for k = 2 to K do
16:     for all j ∈ N do
17:         for all i ∈ N do
18:             if (i, j) ∈ A then
19:                 for all l ∈ L_{i,j}^{k-1} do
20:                     if extend_possible(l, (i, j)) then
21:                         l' = extend(l, (i, j));
22:                         if not dominated(l', \bigcup_{k'=1}^{k} L_{j,j}^{k'}) then
23:                             remove dominated(l', L_{j,j}^k);
24:                             L_j^k = L_j^k \cup \{l'\};
25:                     end if
26:             end if
27:         end if
28:     end for
29: end for
30: end for
31: for all (i, α ∈ A) do
32:     for all k ∈ K do
33:         for all l ∈ L_i^k do
34:             if minCycleCost > c(l) + c_{i,α} then
35:                 minCycleCost = c(l) + c_{i,α};
36:                 bestLabel = l;
37:             end if
38:         end for
39:     end for
40: end for
41: end for
42: end for
3.3.7 All-Pairs like Push Algorithms (APPush)

The All-pairs like push algorithm presented in this section is very similar to the the one presented in Section 3.3.6. The main difference between these two algorithms is the way in which we extend the current labels to create new labels. In APPush algorithms, when constructing labels corresponding to a certain length \( k \), it first decides a start node \( i \) (start node ans source node are different), and extends all paths corresponding to labels in \( L_{i}^{k-1} \) originating in node \( i \) following its outgoing arcs. While in APPull algorithms, when constructing labels corresponding to length \( k \), it first decides an end node \( j \), and extends all paths corresponding to labels in \( L_{j}^{k-1} \), where \((i,j) \in A\). All other functions between these two algorithms remain same. We next present the algorithmic framework.

3.3.8 Strategy to Avoid Inefficient Label Extensions

By using an observation, similar to the those used in the network reduction in Section 3.2, it is possible to avoid some label extensions that never can lead to an optimal cycle. It will not only save an extension but eliminate several labels which would have generated through the current and subsequent extensions.

Assume we are considering extending a label \( l \) at node \( j \) to node \( k \), to construct a new label \( l' \). If label \( l \) has a parent label, denote the node of the parent label as \( i \) (if label \( l \) does not have a parent label, then the technique cannot be applied). In other words, the last arc of the partial path that label \( l \) represents is \((i,j)\) and we consider extending the path with arc \((j,k)\). If there exists a node \( q \) in \( S_{s(j)} \) such that \((i,q) \in A\) and \((q,k) \in A\) and \( c_{iq} + c_{qk} < c_{ij} + c_{jk} \) then we should not extend label \( l \) to node \( k \). Any path that \( l' \) can be extended to can be improved by replacing node \( j \) in the path with node \( q \). We have implemented this strategy as a subroutine in the initial network reduction phase.

The APPush algorithms can be easily modified to solve the SDNCC problems using the same strategy as discussed in Section 3.3.6. We need to replace the accelerating
Algorithm 6 All-pairs-like-push algorithm for SDMCC problems

1: reduction();
2: minCycleCost = \infty
3: bestLabel = NULL
4: for all \( \alpha \in N \) do
5: \( L^k_i = \emptyset \) \( \forall i, k \in 1, \ldots, K \);
6: for all \( i \in N, (\alpha, i) \in A, \text{subset}(\alpha) \neq \text{subset}(i), i > \alpha \) do
7: \( L^1_i = \{(i, c_{\alpha i}, e_{s(\alpha)} + e_{s(i)}, -1)\};
8: if \( (i, \alpha) \in A, i > \alpha \) then
9: if minCycleCost > \( c(l) + c_{i, \alpha} \) then
10: minCycleCost = \( c(l) + c_{i, \alpha} \);
11: bestLabel = \( l \);
12: end if
13: end if
14: end for
15: for \( k = 2 \) to \( K \) do
16: for all \( i \in N, i > \alpha \) do
17: for all \( l \in L^{k-1}_i \) do
18: for all \( (i, j) \in \delta^+ (i), j > \alpha \) do
19: if extend_possible(\( l, (i, j) \)) then
20: \( l' = \text{extend}(l, (i, j)) \);
21: if not_dominated(\( l', \bigcup_{k' = 1}^{k} L^{k'}_{\eta(l')} \)) then
22: remove_dominated(\( l', L^{k}_{\eta(l')} \));
23: \( L^{k}_{\eta(l')} = L^{k}_{\eta(l')} \cup \{l'\} \);
24: end if
25: end if
26: end for
27: end for
28: end for
29: end for
30: for all \( (i, \alpha) \in A, i > \alpha \) do
31: for all \( k \in K \) do
32: for all \( l \in L^{k-1}_i \) do
33: if minCycleCost > \( c(l) + c_{i, \alpha} \) then
34: minCycleCost = \( c(l) + c_{i, \alpha} \);
35: bestLabel = \( l \);
36: end if
37: end for
38: end for
39: end for
40: end for
scheme for minimum cycles with that of negative cycles and extend the feasibility to negative costs.

All these algorithms generate optimal solution for smaller to moderate size instances, however for very large size instances, the performance of the algorithm with respect to the running time deteriorates. We have developed some heuristics in which termination of the algorithm is controlled by changing some parameters of the exact algorithms.

### 3.4 Heuristic Algorithms for Subset Disjoint Minimum Cost Cycles

As discussed earlier the SDMCC problem is NP-complete, and hence, the running time increases exponentially with the size of the problem. As discussed in the computation analysis (Section 3.5), the running time for the problem classes of large size was up to 25 seconds even for the most efficient minimum cycle algorithms. Since, minimum cycle problems are solved repetitively as a subproblem in other complex problems, the high running time is usually not expectable. However, most of the time, instead of finding the optimal cycle, a good quality cycle is sufficient. In this section, we suggest several heuristics by controlling different parameters of the exact algorithms. We have implemented these heuristics only for APPull and APPush algorithms as these were most effective for large size problem-groups, but same concept is applicable for DLA as well as FLA.

#### 3.4.1 Limited Unprocessed Labels Heuristics

All exact algorithms discussed in Section 3.3, maintain a set of labels on each node. These labels may be any combination of subsets in which nodes are divided, and hence, the total number of labels increases exponentially with the number of subsets. The number of labels is stored is directly proportional to the running time of the algorithm. We suggest controlling these parameters by maintaining a limited number of labels stored for future processing at each label. More specifically, if the limit on each node is \( N \), and the newly created label takes the total number to \( N + 1 \), we scan the list of
unprocessed label and delete the one which has least probability of converting into the optimal label. Hence, in all the exact algorithms, we insert the following line while updating the set of labels at each node:

\[
\text{IF } |L_j| > N \text{ THEN remove the label of maximum cost from } L_j.
\]

We discuss the effectiveness of these algorithms in the computational section. Since, this heuristic is required only for problem-groups of large size for which the exact algorithms took more than a second, we perform tests only for those problem groups.

### 3.4.2 Limited Cycle Length Heuristics

The second heuristics is based on the idea of cycle length, and therefore, applicable only to all-pairs like algorithms. In most of the neighborhood search algorithms, only two-exchanges are required, which corresponds to the cycles of length two. Other algorithms first try to find two exchanges, and thereafter, search multiple exchanges but within fixed length \( m \). In the proposed heuristic, we use this logic to maintain the cycles only up to a fixed length \( m \), instead of \( M \), where \( m < M \). For APPull and APPush algorithms, we reduce the number of labels stored for processing by limiting the length of the path which is converted to cycle. As soon as paths of length \( k - 1 \) are found for each source, we check the cycle criteria and don’t extend those labels any more. In the algorithms discussed in Sections 3.3.6 and 3.3.7, we substitute the “for-loop” extending the path one by one to \( K \) by following line: \( \text{for all } k = 2, 3, ..., m \text{ do} \) In the cycle check phase, we also check only up to length \( m \) as there is nothing stored in sets beyond that. The computational results is shown for all four problem groups of large size which had high running time in exact approaches.

### 3.4.3 Limited Cycle Number Heuristics

Our third heuristic approach is most suitable for the neighborhood search strategy in which we are concerned only with finding the negative cycle without giving much weight to its quality. It can be applied to all exact algorithms proposed in this chapter, however, we show the computational results only for APPull and APPush algorithms.
We set a limit $m$ on the number of cycles discovered by the algorithm. In the case of SDNCC problems, we use the extra clause that as soon as $m$ negative cycles are found, we stop the algorithm. The solution obtained is the least cost cycle among all discovered by the algorithm among $m$ cycles. More precisely, whenever we update the incumbent cycle with a new one. More precisely we add following lines in the code:

```
num_cycle = num_cycle + 1  
if (num_cycle > m) then STOP
```

It should be noted that in the case of SDMCC problems, if the network doesn’t contain any negative cycle, it is equivalent to an exact algorithm in which we find the subset disjoint minimum positive cost cycle. The algorithm keeps updating the incumbent cycle with better till it finds a non-existent negative cycle. For SDNCC problems, the algorithms stops as soon as $m$ cycles are found. So, the solution quality depends on the number of feasible negative cycles in the network. If the network contains very few SDNCCs, the quality of solutions obtained would be much better.

This heuristic approach is most suited for those problem classes in which master problem needs some set of good but not optimal cycles through subproblems which can be modelled as SDMCC problems. The users can play with the parameter $m$ to see the affect of global solutions. We discuss the computational details in Section 3.5.

### 3.5 Computational Analysis

The performance of all exact and heuristics algorithms discussed in this chapter are compared on a set of benchmark problems discussed in [31]. These benchmark problems were generated by considering different instances of the capacitated minimum spanning tree problems in paper by Ahuja et al. [6]. The paper solves the capacitated minimum spanning tree problems using very-large scale neighborhood search, in which the subset-disjoint negative cycle problem is solved as the subproblem. We take each instance of the spanning tree problem from [6], and group all its subproblems. These groups of subproblems serve as the input for comparing our algorithms. The instances for capacitated minimum spanning tree problems were generated in two ways:
random: nodes on the grid of a large square and root node at the center of square, arcs cost linearly related to their Euclidean distance. The complete details along with the benchmark problems can be found in [13]. We test all our algorithms proposed on these group of problems. We solve the all problems in a group (subproblems generated by an instance of capacitated minimum cycle problem) and present different statistics such as the arc density, percentage of negative arcs, average running time, average maximum labels stored. Each group of the problems is represented by a unique id and the number of problems solved in it. For example, a problems with id "44-10" stands for the group number 44 with 10 subset disjoint minimum cost cycle instances. As discussed earlier percentage of negative arcs help in selected of acceleration strategy in the negative cycle version of the problem. The number of labels stores in an algorithm is a direct indication of it memory requirement. The arc density is calculated with respect to the ideal number of full arcs which can be present in a perfectly dense network. Ideal arcs exclude the arcs that are between the nodes that belong to the same subsets. The maximum number of arcs in a perfectly dense network can be 

\[ n(n - 1) - \sum_{k=1}^{K} |S_k||S_k| - 1. \]

The negative arc as well as total arc densities have been mentioned with respect to the maximum number of ideal arcs.

For exact algorithms, we report the number of nodes, number of subsets (multiple entries if number of subsets are different), total arcs density, negative arcs density, average running time for each group, and average of the highest number of the labels stored for each group. In the case of heuristics, we also report the average deviation from the optimal solution along with the value of parameter controlled. All network reduction and acceleration strategies have been used in each algorithm. These are programmed in C++ using efficient data structures, and tested on a 2.39 GHz PC with 1.99 GB of RAM.

We first test all four exact SDMCC algorithms on three classes of problem instances, classified as per the number of subsets in which all nodes are divided.
The comparative results for the small size problems are given in Table 3-1, for the medium size problems in Table 3-2, and for the large size problems in Table 3-3. In all the tables "P" denotes problem-groups, "Sets" denotes the number of subsets in the graph, "Nodes" denotes the number of nodes in the graph, "Arcs" denotes the arc density in percentage, "Neg. Arcs" denotes the density of negative arcs in the graph, "AML" denotes the average maximum number of labels stored by algorithms, and "time" denotes the time consumed in seconds.

In Table 3-1, we compare all algorithms on the small size instances. The problems have been sorted in decreasing order of number of subsets, and for the same number of subsets, in the increasing order of number of nodes. In all these instances, from time as well as memory perspective, the dynamic labeling algorithm performed much better than the fixed labeling algorithm. For 17 out of 20 instances, the time required by DLA was almost half of that required by FLA. The average maximum number of labels stored by FLA was 1.5 - 2 times of average maintained by DLA. It is not very intuitive in the first glance as in the FLA, we store labels for each node, only if it is certain that the label would never be removed through dominance test. In other words, at each node, we maintain the list of processed labels which are fixed. During the label extension step, we insert all newly created labels, only in the set of unprocessed labels. We take out labels one by one from this list and check if it is dominated. It should be noted that, when the label is inserted into the unprocessed label set, it is not dominated any of the label fixed at the corresponding node. However, by the time that label is removed from the unprocessed label set for processing, it may be dominated by the new labels created and fixed at that node. However, in the DLA, the labels stored in set of labels maintained at each node may get dominated by some future labels, and hence, may be removed. In the removal operation, all labels from the unprocessed label-set as well as from the node-specific-label-set, which are dominated by the newly created label are removed. So, although we may have to perform label removal and label entry steps multiple time,
overall number of labels stored are much lower than FLA. Since, the number of labels stored in the unprocessed label set is less, total number of "first()" operation is low, which mainly results in the lower solution time. The performance of both "all-pair like algorithms" was much better than DLA (which is better than FLA). The main difference between these algorithms is that in "all-pair like algorithms", we sequentially find the path of increasing lengths. While in FLA and DLA, we extend the labels in a predefined ordering, disregarding the length of the paths. The number of average maximum labels stored was not very different in DLA, APPull, and APPush. However, the time taken by DLA was much higher (almost double) than APPull and APPush algorithms. The number of labels stored for APPush and APPull algorithms were exactly same, as in both algorithms we use the strategy of DLA and keep adding/removing labels in the set at each node. The time taken by these algorithms is not very different with APPush algorithm performing better in most instances. APPush algorithm is faster in 12 problem-groups, same in six groups, while worse in one of 20 problem-groups. The basic concept of both algorithms is same and difference comes only in the way we update and extend labels. Therefore, for this problem-group, we can conclude that APPush algorithms is best for these problem classes.

Table 3-2 compares the same set of algorithms on problem classes of medium size. We consider the same set of nodes, and increase the number of subsets (14-17) in which these are classified. The comparative behavior of all four exact algorithms is almost same as we have discussed for problem groups of small size. The difference in DLA and FLA with regard to the number of labels stored as well as solution time, becomes more prominent as the absolute difference is very high (although difference in ratio is same). All but four problems were solved within fraction of seconds. These four groups were largest (199) with respect to the number of nodes. The comparative performance of APPull algorithm was much better in this problem class than the smaller class. APPull algorithm performed better than APPush in five out of 20 problem-groups,
same in five groups, while worse in four groups. Hence, any of these two algorithms would work fine for this class.

Table 3-3 displays the results for the large problem groups. The number of nodes vary from 80 to 199, and number of subsets from 17 to 21. The arc density is also higher from last two problem classes. compares the same set of algorithms on problem classes of medium size. In this case too, we observe the same pattern of performance among four exact algorithms. However, the difference in solution between "DLA-FLA", and "APPull-APPush" algorithms was very significant. For example, one of the problem-group took 137.36 seconds for DLA, 255.88 seconds for FLA, 20.97 seconds for APPull, and 32.77 seconds for APPush. It clearly shows the efficiency of APPull and APPush algorithms. Among APPull and APPush algorithms, the APPull algorithms performed better than APPush in all problem-groups. This behavior is quite interesting as for small instances APPush algorithms has better running time. So, we can conclude that APPull algorithms are best for these classes of problems. However, even the most efficient algorithm took more than 20 seconds for two problem-groups, and more than 10 seconds for one group. For most of those problems which solve SDMCC repetitively as sub-problems, this running time is unacceptable. So, we developed some heuristics, and we discuss these results after the analysis of SDNCC algorithms.

We discuss the results of SDNCC problems for problem-groups of small size in Table 3-4. All algorithms performed very efficiently in all these instances generating optimum results in less than 0.07 seconds. The memory requirement was also very low storing at most 1300 labels. The pattern between DLA and FLA is same as it was in the SDMCC problem as shown in Table 3-1. Among APPull and APPush algorithms, APPush performed better in all instances from time perspective. However, the pattern between DLA and APPull algorithm was opposite of SDMCC results. The DLA performed consistently better than APPull algorithms from time perspective, and the number of labels stored were in the same range (slightly more in DLA). Therefore,
Table 3-1. Comparative analysis of exact SDMCC algorithms: small instances.

<table>
<thead>
<tr>
<th>P sets</th>
<th>nodes</th>
<th>Arcs</th>
<th>Neg. Arcs</th>
<th>DLA</th>
<th>FLA</th>
<th>AP Pull</th>
<th>AP Push</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>AML Time</td>
<td>AML Time</td>
<td>AML Time</td>
<td>AML Time</td>
<td>AML Time</td>
</tr>
<tr>
<td>33-12</td>
<td>13</td>
<td>99</td>
<td>69.81</td>
<td>4,026 0.25</td>
<td>7,058 0.55</td>
<td>3,820 0.18</td>
<td>3,820 0.16</td>
</tr>
<tr>
<td>35-8</td>
<td>12,13</td>
<td>99</td>
<td>67.82</td>
<td>3,274 0.20</td>
<td>5,056 0.44</td>
<td>3,217 0.16</td>
<td>3,217 0.13</td>
</tr>
<tr>
<td>30-15</td>
<td>12</td>
<td>99</td>
<td>75.01</td>
<td>3,650 0.24</td>
<td>6,349 0.55</td>
<td>3,507 0.17</td>
<td>3,507 0.15</td>
</tr>
<tr>
<td>27-9</td>
<td>12</td>
<td>99</td>
<td>69.28</td>
<td>2,850 0.18</td>
<td>4,475 0.40</td>
<td>2,745 0.14</td>
<td>2,745 0.12</td>
</tr>
<tr>
<td>39-16</td>
<td>11</td>
<td>199</td>
<td>62.28</td>
<td>5,206 0.74</td>
<td>8,808 1.70</td>
<td>4,880 0.73</td>
<td>4,880 0.59</td>
</tr>
<tr>
<td>42-7</td>
<td>11</td>
<td>199</td>
<td>68.25</td>
<td>7,580 1.19</td>
<td>13,526 2.80</td>
<td>7,267 1.03</td>
<td>7,267 0.84</td>
</tr>
<tr>
<td>45-11</td>
<td>11</td>
<td>199</td>
<td>75.25</td>
<td>7,324 1.03</td>
<td>13,487 2.57</td>
<td>7,078 0.89</td>
<td>7,078 0.74</td>
</tr>
<tr>
<td>48-14</td>
<td>11</td>
<td>199</td>
<td>76.25</td>
<td>5,791 0.95</td>
<td>10,328 2.30</td>
<td>5,582 0.84</td>
<td>5,582 0.68</td>
</tr>
<tr>
<td>2-11</td>
<td>10,11</td>
<td>40</td>
<td>85.95</td>
<td>731 0.02</td>
<td>987 0.03</td>
<td>644 0.02</td>
<td>644 0.00</td>
</tr>
<tr>
<td>11-8</td>
<td>10,11</td>
<td>40</td>
<td>85.96</td>
<td>526 0.02</td>
<td>798 0.02</td>
<td>500 0.01</td>
<td>500 0.01</td>
</tr>
<tr>
<td>5-6</td>
<td>10</td>
<td>40</td>
<td>85.71</td>
<td>688 0.03</td>
<td>810 0.02</td>
<td>660 0.02</td>
<td>660 0.00</td>
</tr>
<tr>
<td>8-11</td>
<td>10</td>
<td>40</td>
<td>85.71</td>
<td>814 0.03</td>
<td>1,001 0.02</td>
<td>743 0.01</td>
<td>743 0.01</td>
</tr>
<tr>
<td>18-14</td>
<td>10</td>
<td>80</td>
<td>83.72</td>
<td>1,272 0.04</td>
<td>1,797 0.09</td>
<td>1,130 0.04</td>
<td>1,130 0.03</td>
</tr>
<tr>
<td>24-14</td>
<td>10</td>
<td>80</td>
<td>83.72</td>
<td>864 0.02</td>
<td>1,385 0.06</td>
<td>781 0.02</td>
<td>781 0.02</td>
</tr>
<tr>
<td>15-12</td>
<td>10</td>
<td>80</td>
<td>81.92</td>
<td>745 0.02</td>
<td>1,196 0.04</td>
<td>677 0.02</td>
<td>677 0.02</td>
</tr>
<tr>
<td>21-11</td>
<td>10</td>
<td>80</td>
<td>80.30</td>
<td>528 0.01</td>
<td>802 0.03</td>
<td>483 0.02</td>
<td>483 0.01</td>
</tr>
<tr>
<td>3-5</td>
<td>8</td>
<td>40</td>
<td>81.40</td>
<td>283 0.01</td>
<td>352 0.02</td>
<td>254 0.00</td>
<td>254 0.01</td>
</tr>
<tr>
<td>6-9</td>
<td>8</td>
<td>40</td>
<td>81.40</td>
<td>375 0.01</td>
<td>449 0.01</td>
<td>308 0.00</td>
<td>308 0.00</td>
</tr>
<tr>
<td>9-11</td>
<td>8,9</td>
<td>40</td>
<td>81.78</td>
<td>465 0.01</td>
<td>599 0.01</td>
<td>409 0.01</td>
<td>409 0.01</td>
</tr>
<tr>
<td>12-7</td>
<td>7,8</td>
<td>40</td>
<td>78.34</td>
<td>188 0.01</td>
<td>244 0.01</td>
<td>163 0.00</td>
<td>163 0.00</td>
</tr>
<tr>
<td>P sets nodes</td>
<td>Arcs</td>
<td>Neg. Arcs</td>
<td>DLA AML Time</td>
<td>FLA AML Time</td>
<td>APPull AML Time</td>
<td>APPush AML Time</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>-----------</td>
<td>--------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>14-19</td>
<td>16,17</td>
<td>80</td>
<td>90.38</td>
<td>7.04</td>
<td>9,145</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18,989</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9,046</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9,046</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>23-16</td>
<td>16,17</td>
<td>80</td>
<td>90.39</td>
<td>7.78</td>
<td>16,711</td>
<td>0.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22,316</td>
<td>1.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15,992</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15,992</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td>20-16</td>
<td>16</td>
<td>80</td>
<td>90.36</td>
<td>7.39</td>
<td>12,557</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23,443</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12,492</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12,492</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>32-23</td>
<td>15,16</td>
<td>99</td>
<td>71.11</td>
<td>3.79</td>
<td>10,031</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16,533</td>
<td>1.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9,572</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9,572</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>26-7</td>
<td>15</td>
<td>99</td>
<td>70.48</td>
<td>4.97</td>
<td>8,370</td>
<td>0.61</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13,479</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8,323</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8,323</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>38-17</td>
<td>14,15</td>
<td>199</td>
<td>64.84</td>
<td>2.08</td>
<td>26,152</td>
<td>3.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>52,865</td>
<td>8.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25,167</td>
<td>2.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25,167</td>
<td>2.03</td>
<td></td>
</tr>
<tr>
<td>29-14</td>
<td>14</td>
<td>99</td>
<td>65.95</td>
<td>4.57</td>
<td>9,091</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14,789</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8,906</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8,906</td>
<td>0.29</td>
<td></td>
</tr>
<tr>
<td>41-7</td>
<td>14</td>
<td>199</td>
<td>74.09</td>
<td>1.99</td>
<td>12,356</td>
<td>2.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25,172</td>
<td>5.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12,116</td>
<td>1.64</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12,116</td>
<td>1.50</td>
<td></td>
</tr>
<tr>
<td>44-10</td>
<td>14</td>
<td>199</td>
<td>72.79</td>
<td>1.53</td>
<td>22,715</td>
<td>5.53</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>40,040</td>
<td>11.85</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22,099</td>
<td>2.70</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22,099</td>
<td>2.80</td>
<td></td>
</tr>
<tr>
<td>47-6</td>
<td>14</td>
<td>199</td>
<td>76.49</td>
<td>1.59</td>
<td>15,354</td>
<td>3.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31,840</td>
<td>7.80</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15,086</td>
<td>1.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15,086</td>
<td>1.72</td>
<td></td>
</tr>
<tr>
<td>1-5</td>
<td>14</td>
<td>40</td>
<td>90.71</td>
<td>11.90</td>
<td>2,684</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,347</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2,622</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2,622</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>4-4</td>
<td>14</td>
<td>40</td>
<td>90.48</td>
<td>12.55</td>
<td>2,124</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,174</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2,071</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2,071</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>7-10</td>
<td>14</td>
<td>40</td>
<td>90.48</td>
<td>17.25</td>
<td>1,985</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,139</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,961</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,961</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>10-6</td>
<td>14</td>
<td>40</td>
<td>90.48</td>
<td>14.77</td>
<td>1,634</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3,162</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,595</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,595</td>
<td>0.02</td>
<td></td>
</tr>
</tbody>
</table>
Table 3.3. Comparative analysis of exact SDMCC algorithms: large instances

<table>
<thead>
<tr>
<th></th>
<th>sets</th>
<th>nodes</th>
<th>Arcs</th>
<th>Neg. Arcs</th>
<th>DLA AML</th>
<th>DLA Time</th>
<th>FLA AML</th>
<th>FLA Time</th>
<th>APPull AML</th>
<th>APPull Time</th>
<th>APPush AML</th>
<th>APPush Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>13-11</td>
<td>21</td>
<td>80</td>
<td>92.87</td>
<td>9.24</td>
<td>15,172</td>
<td>1.06</td>
<td>35,739</td>
<td>2.48</td>
<td>15,094</td>
<td>0.34</td>
<td>15,094</td>
<td>0.40</td>
</tr>
<tr>
<td>19-27</td>
<td>21</td>
<td>80</td>
<td>92.92</td>
<td>9.19</td>
<td>37,351</td>
<td>2.21</td>
<td>61,055</td>
<td>4.14</td>
<td>35,126</td>
<td>0.65</td>
<td>35,126</td>
<td>0.83</td>
</tr>
<tr>
<td>16-9</td>
<td>20,21</td>
<td>80</td>
<td>92.84</td>
<td>9.61</td>
<td>21,350</td>
<td>1.57</td>
<td>38,134</td>
<td>3.12</td>
<td>20,772</td>
<td>0.46</td>
<td>20,772</td>
<td>0.56</td>
</tr>
<tr>
<td>25-53</td>
<td>19,21</td>
<td>99</td>
<td>77.44</td>
<td>6.02</td>
<td>137,256</td>
<td>72.18</td>
<td>219,269</td>
<td>117.29</td>
<td>136,548</td>
<td>6.15</td>
<td>136,548</td>
<td>12.17</td>
</tr>
<tr>
<td>22-11</td>
<td>20</td>
<td>80</td>
<td>92.68</td>
<td>10.46</td>
<td>50,692</td>
<td>3.20</td>
<td>81,280</td>
<td>5.67</td>
<td>50,225</td>
<td>0.80</td>
<td>50,225</td>
<td>1.03</td>
</tr>
<tr>
<td>31-17</td>
<td>19</td>
<td>99</td>
<td>69.53</td>
<td>4.49</td>
<td>23,535</td>
<td>2.19</td>
<td>41,201</td>
<td>4.42</td>
<td>23,217</td>
<td>0.76</td>
<td>23,217</td>
<td>0.91</td>
</tr>
<tr>
<td>34-18</td>
<td>18,19</td>
<td>99</td>
<td>71.81</td>
<td>4.13</td>
<td>18,265</td>
<td>1.52</td>
<td>33,420</td>
<td>3.26</td>
<td>17,605</td>
<td>0.57</td>
<td>17,605</td>
<td>0.65</td>
</tr>
<tr>
<td>28-25</td>
<td>18,19</td>
<td>99</td>
<td>66.75</td>
<td>5.82</td>
<td>51,336</td>
<td>5.88</td>
<td>79,140</td>
<td>10.60</td>
<td>51,042</td>
<td>1.49</td>
<td>51,042</td>
<td>1.93</td>
</tr>
<tr>
<td>46-21</td>
<td>18</td>
<td>199</td>
<td>73.51</td>
<td>2.65</td>
<td>140,916</td>
<td>137.36</td>
<td>233,972</td>
<td>255.88</td>
<td>139,416</td>
<td>20.97</td>
<td>139,416</td>
<td>32.77</td>
</tr>
<tr>
<td>43-22</td>
<td>18</td>
<td>199</td>
<td>76.58</td>
<td>2.14</td>
<td>92,281</td>
<td>47.42</td>
<td>131,579</td>
<td>87.29</td>
<td>91,711</td>
<td>10.45</td>
<td>91,711</td>
<td>14.66</td>
</tr>
<tr>
<td>37-17</td>
<td>18</td>
<td>199</td>
<td>73.53</td>
<td>2.65</td>
<td>37,523</td>
<td>6.90</td>
<td>87,116</td>
<td>17.97</td>
<td>36,475</td>
<td>3.00</td>
<td>36,475</td>
<td>3.27</td>
</tr>
<tr>
<td>40-54</td>
<td>17,18</td>
<td>199</td>
<td>71.53</td>
<td>3.47</td>
<td>171,963</td>
<td>132.81</td>
<td>283,052</td>
<td>264.11</td>
<td>170,941</td>
<td>20.95</td>
<td>170,941</td>
<td>34.25</td>
</tr>
</tbody>
</table>
we can safely conclude that DLA is the best algorithm to solve this class of instances while finding SDNCC.

In Table 3-5, we show the output of all algorithms on problems-groups of intermediate size. All the algorithms were very effective in these classes of problems as well solving 12 out of 14 groups within 0.015 seconds and two groups within 0.15 seconds. The comparative behavior of DLA and FLA remains same as for all other groups. The APPush algorithms found optimal cycles much quicker than (almost 1.5 times) APPull algorithms for all instances. However, among DLA and APPush algorithms, there was no clear difference in performance. In eight out of 14 problem-groups DLA was quicker than APPush algorithms, while in five groups APPush algorithms outperformed DLA. There was tie in one group, which should be counted towards APPush algorithms if memory requirement is our slightest concern. The number of labels stored was consistently lower in APPush algorithms.

In Table 3-6, we analyse the performance of each exact algorithm for SDNCC problems on large size instances. There was no change in the pattern of DLA and FLA from our earlier analysis. The APPush algorithms again outperformed APPull algorithms in all problem groups. The most interesting observation was the shift in the behavior of DLA and APPush algorithms. In these problem groups, DLA was completely overshadowed by APPush algorithms and performed better in only two out of 14 instances. In those two instances too, the time difference was not very significant, with APPush algorithm storing much less number of labels. In the most difficult instance of this class, DLA algorithm took 24.5 seconds to find the optimal results, however, the APPush algorithm generated it in 5.4 seconds. It shows that if the problem size is not small to medium range, DLA is a better approach but if the size is very large, APPush algorithms perform the best.

We now test the heuristics approaches suggested in Section 3.4, which control three parameters of the exact algorithms: (i) Number of unprocessed lables, (ii)
Table 3-4. Comparative analysis of exact SDNCC algorithms: small instances

<table>
<thead>
<tr>
<th>P sets</th>
<th>nodes</th>
<th>Arcs</th>
<th>Neg. Arcs</th>
<th>DLA AML Time</th>
<th>FLA AML Time</th>
<th>APPull AML Time</th>
<th>APPush AML Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>33-12</td>
<td>13</td>
<td>99</td>
<td>69.81</td>
<td>2.64</td>
<td>1,021 0.017</td>
<td>1,162 0.030</td>
<td>943 0.034</td>
</tr>
<tr>
<td>35-8</td>
<td>12,13</td>
<td>99</td>
<td>67.82</td>
<td>2.83</td>
<td>548 0.012</td>
<td>632 0.015</td>
<td>526 0.025</td>
</tr>
<tr>
<td>30-15</td>
<td>12</td>
<td>99</td>
<td>75.01</td>
<td>3.68</td>
<td>546 0.015</td>
<td>616 0.022</td>
<td>512 0.025</td>
</tr>
<tr>
<td>27-9</td>
<td>12</td>
<td>99</td>
<td>69.28</td>
<td>2.79</td>
<td>186 0.012</td>
<td>204 0.019</td>
<td>165 0.026</td>
</tr>
<tr>
<td>39-16</td>
<td>11</td>
<td>199</td>
<td>62.28</td>
<td>1.41</td>
<td>477 0.062</td>
<td>544 0.059</td>
<td>407 0.144</td>
</tr>
<tr>
<td>42-7</td>
<td>11</td>
<td>199</td>
<td>68.25</td>
<td>1.58</td>
<td>1,233 0.083</td>
<td>1,539 0.087</td>
<td>1,151 0.172</td>
</tr>
<tr>
<td>45-11</td>
<td>11</td>
<td>199</td>
<td>199</td>
<td>1.24</td>
<td>1,369 0.077</td>
<td>1,774 0.090</td>
<td>1,264 0.151</td>
</tr>
<tr>
<td>48-14</td>
<td>11</td>
<td>199</td>
<td>76.25</td>
<td>1.1</td>
<td>1,059 0.074</td>
<td>1,302 0.079</td>
<td>985 0.141</td>
</tr>
<tr>
<td>2-11</td>
<td>10,11</td>
<td>40</td>
<td>85.95</td>
<td>9.59</td>
<td>247 0.000</td>
<td>273 0.004</td>
<td>226 0.001</td>
</tr>
<tr>
<td>11-8</td>
<td>10,11</td>
<td>40</td>
<td>85.96</td>
<td>9.7</td>
<td>161 0.002</td>
<td>159 0.006</td>
<td>149 0.004</td>
</tr>
<tr>
<td>5-6</td>
<td>10</td>
<td>40</td>
<td>85.71</td>
<td>9.99</td>
<td>132 0.003</td>
<td>119 0.005</td>
<td>115 0.008</td>
</tr>
<tr>
<td>8-11</td>
<td>10</td>
<td>40</td>
<td>85.71</td>
<td>11.47</td>
<td>262 0.001</td>
<td>266 0.009</td>
<td>239 0.009</td>
</tr>
<tr>
<td>18-14</td>
<td>10</td>
<td>80</td>
<td>83.72</td>
<td>5.82</td>
<td>446 0.006</td>
<td>399 0.009</td>
<td>370 0.008</td>
</tr>
<tr>
<td>24-14</td>
<td>10</td>
<td>80</td>
<td>83.72</td>
<td>8.69</td>
<td>399 0.003</td>
<td>504 0.008</td>
<td>329 0.010</td>
</tr>
<tr>
<td>15-12</td>
<td>10</td>
<td>80</td>
<td>81.92</td>
<td>5.9</td>
<td>230 0.006</td>
<td>250 0.007</td>
<td>194 0.009</td>
</tr>
<tr>
<td>21-11</td>
<td>10</td>
<td>80</td>
<td>80.3</td>
<td>5.51</td>
<td>341 0.004</td>
<td>331 0.004</td>
<td>310 0.010</td>
</tr>
<tr>
<td>3-5</td>
<td>8</td>
<td>40</td>
<td>81.4</td>
<td>9.89</td>
<td>92 0.003</td>
<td>98 0.000</td>
<td>84 0.006</td>
</tr>
<tr>
<td>6-9</td>
<td>8</td>
<td>40</td>
<td>81.4</td>
<td>8.7</td>
<td>96 0.000</td>
<td>92 0.005</td>
<td>77 0.003</td>
</tr>
<tr>
<td>9-11</td>
<td>8,9</td>
<td>40</td>
<td>81.78</td>
<td>11.87</td>
<td>234 0.004</td>
<td>251 0.007</td>
<td>190 0.006</td>
</tr>
<tr>
<td>12-7</td>
<td>7,8</td>
<td>40</td>
<td>78.34</td>
<td>10.09</td>
<td>89 0.002</td>
<td>85 0.000</td>
<td>78 0.002</td>
</tr>
<tr>
<td>P sets</td>
<td>nodes</td>
<td>Arcs</td>
<td>Neg. Arcs</td>
<td>DLA AML Time</td>
<td>AML Time</td>
<td>FLA AML Time</td>
<td>AML Time</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>------</td>
<td>-----------</td>
<td>--------------</td>
<td>----------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>14-19</td>
<td>16,17</td>
<td>80</td>
<td>90.38</td>
<td>7.04</td>
<td>2,171</td>
<td>0.020</td>
<td>3,381</td>
</tr>
<tr>
<td>23-16</td>
<td>16,17</td>
<td>80</td>
<td>90.39</td>
<td>7.78</td>
<td>2,163</td>
<td>0.022</td>
<td>2,256</td>
</tr>
<tr>
<td>20-16</td>
<td>16</td>
<td>80</td>
<td>90.36</td>
<td>7.39</td>
<td>1,520</td>
<td>0.018</td>
<td>1,834</td>
</tr>
<tr>
<td>32-23</td>
<td>15,16</td>
<td>99</td>
<td>71.11</td>
<td>3.79</td>
<td>1,308</td>
<td>0.018</td>
<td>1,466</td>
</tr>
<tr>
<td>26-7</td>
<td>15</td>
<td>99</td>
<td>70.48</td>
<td>4.97</td>
<td>525</td>
<td>0.018</td>
<td>507</td>
</tr>
<tr>
<td>38-17</td>
<td>14,15</td>
<td>199</td>
<td>64.84</td>
<td>2.08</td>
<td>3,485</td>
<td>0.128</td>
<td>6,421</td>
</tr>
<tr>
<td>29-14</td>
<td>14</td>
<td>99</td>
<td>65.95</td>
<td>4.57</td>
<td>2,754</td>
<td>0.026</td>
<td>3,162</td>
</tr>
<tr>
<td>41-7</td>
<td>14</td>
<td>199</td>
<td>74.09</td>
<td>1.99</td>
<td>1,007</td>
<td>0.092</td>
<td>1,150</td>
</tr>
<tr>
<td>44-10</td>
<td>14</td>
<td>199</td>
<td>72.79</td>
<td>1.53</td>
<td>3,480</td>
<td>0.159</td>
<td>4,300</td>
</tr>
<tr>
<td>47-6</td>
<td>14</td>
<td>199</td>
<td>76.49</td>
<td>1.59</td>
<td>1,994</td>
<td>0.094</td>
<td>2,558</td>
</tr>
<tr>
<td>1-5</td>
<td>14</td>
<td>40</td>
<td>90.71</td>
<td>11.9</td>
<td>605</td>
<td>0.003</td>
<td>603</td>
</tr>
<tr>
<td>4-4</td>
<td>14</td>
<td>40</td>
<td>90.48</td>
<td>12.55</td>
<td>524</td>
<td>0.004</td>
<td>513</td>
</tr>
<tr>
<td>7-10</td>
<td>14</td>
<td>40</td>
<td>90.48</td>
<td>17.25</td>
<td>1,098</td>
<td>0.013</td>
<td>1,157</td>
</tr>
<tr>
<td>10-6</td>
<td>14</td>
<td>40</td>
<td>90.48</td>
<td>14.77</td>
<td>703</td>
<td>0.013</td>
<td>859</td>
</tr>
</tbody>
</table>
Table 3-6. Comparative analysis of exact SDNCC algorithms: large instances

<table>
<thead>
<tr>
<th>P sets</th>
<th>nodes</th>
<th>Arc -ve</th>
<th>DLA AML</th>
<th>DLA Time</th>
<th>FLA AML</th>
<th>FLA Time</th>
<th>APPull AML</th>
<th>APPull Time</th>
<th>APPush AML</th>
<th>APPush Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>13-11</td>
<td>21</td>
<td>80</td>
<td>92.87</td>
<td>9.24</td>
<td>2,299</td>
<td>0.026</td>
<td>2,432</td>
<td>0.040</td>
<td>2,264</td>
<td>0.036</td>
</tr>
<tr>
<td>19-27</td>
<td>21</td>
<td>80</td>
<td>92.92</td>
<td>9.19</td>
<td>11,717</td>
<td>0.177</td>
<td>11,352</td>
<td>0.252</td>
<td>10,748</td>
<td>0.094</td>
</tr>
<tr>
<td>16-9</td>
<td>20,21</td>
<td>80</td>
<td>92.84</td>
<td>9.61</td>
<td>1,956</td>
<td>0.023</td>
<td>2,375</td>
<td>0.031</td>
<td>1,861</td>
<td>0.026</td>
</tr>
<tr>
<td>25-53</td>
<td>19,21</td>
<td>99</td>
<td>77.44</td>
<td>6.02</td>
<td>40,011</td>
<td>5.791</td>
<td>59,759</td>
<td>9.677</td>
<td>39,643</td>
<td>0.893</td>
</tr>
<tr>
<td>22-11</td>
<td>20</td>
<td>80</td>
<td>92.68</td>
<td>10.46</td>
<td>4,797</td>
<td>0.057</td>
<td>6,983</td>
<td>0.099</td>
<td>4,529</td>
<td>0.048</td>
</tr>
<tr>
<td>31-17</td>
<td>19</td>
<td>99</td>
<td>69.53</td>
<td>4.49</td>
<td>2,921</td>
<td>0.046</td>
<td>3,318</td>
<td>0.065</td>
<td>2,853</td>
<td>0.062</td>
</tr>
<tr>
<td>34-18</td>
<td>18,19</td>
<td>99</td>
<td>71.81</td>
<td>4.13</td>
<td>1,595</td>
<td>0.028</td>
<td>1,899</td>
<td>0.054</td>
<td>1,533</td>
<td>0.051</td>
</tr>
<tr>
<td>28-25</td>
<td>18,19</td>
<td>99</td>
<td>66.75</td>
<td>5.82</td>
<td>21,174</td>
<td>0.632</td>
<td>24,746</td>
<td>1.017</td>
<td>20,920</td>
<td>0.214</td>
</tr>
<tr>
<td>46-21</td>
<td>18</td>
<td>199</td>
<td>73.51</td>
<td>2.65</td>
<td>25,290</td>
<td>1.728</td>
<td>36,019</td>
<td>2.893</td>
<td>24,929</td>
<td>0.864</td>
</tr>
<tr>
<td>43-22</td>
<td>18</td>
<td>199</td>
<td>76.58</td>
<td>2.14</td>
<td>16,205</td>
<td>0.786</td>
<td>18,778</td>
<td>1.016</td>
<td>16,043</td>
<td>0.509</td>
</tr>
<tr>
<td>37-17</td>
<td>18</td>
<td>199</td>
<td>73.53</td>
<td>2.65</td>
<td>2,430</td>
<td>0.134</td>
<td>3,166</td>
<td>0.152</td>
<td>2,320</td>
<td>0.320</td>
</tr>
<tr>
<td>40-54</td>
<td>17,18</td>
<td>199</td>
<td>71.53</td>
<td>3.47</td>
<td>133,621</td>
<td>24.586</td>
<td>185,563</td>
<td>37.862</td>
<td>132,909</td>
<td>3.064</td>
</tr>
</tbody>
</table>

AML Time
Maximum length of cycles, and (iii) Total number of cycles found. First and third approaches can be applied to any of the four exact algorithms, however, second approach is applicable only to APPull and APPush algorithms. While analysing the results generated by exact algorithms for SDMCC problems, we realized that APPull and APPush algorithms performed much better than DLA and FLA for the large size problem groups. So, we decided to develop heuristics with respect to only the all-pairs-like algorithms. In out computational tables we refer to there heuristics as LUL, LCL, and LCN. First we discuss the results on SDMCC large size instances. In Tables 3-7, and 3-8, "P" denotes the problem-groups, "Opt." denotes the optimal solution, "Dev." denotes the deviation of heuristic solution from the optimal, and "Time" denotes the running time in seconds.

For the LUL heuristics, we fixed the limit of unprocessed labels to 10, 50, and 1000 for APPull and APPush algorithms. The solution quality and running time, as expected improved with increase in the number of labels allowed. As we increased the label limit from 10 to 1000, the gap between the heuristic solution and optimal solution improved from 30% to almost 1% while time increased from one second to three second. In the most difficult instance, the time taken to generate optimal solution, for APPull and APPush algorithms, was 20 seconds and 34 seconds which improved to 3.74 seconds and 3.60 seconds respectively. The optimality gap was around 2%. Both generated optimal solution for one problem-group. Among APPull and APPush algorithms with LUL, APPull generated solutions with better gap while APPush was faster specially with lower number of labels. For LCL, we varied the limit on the maximum cycle length to 2, 4, and 8. Both APPull and APPush version of algorithms generated poor solutions with cycle length two, which denotes a simple swapping. It is due to the fact that average optimal cycle lengths in all problems group vary from eight to 10. With the cycle length limit 8, the running time was very high considering that it is a heuristic. The LUL outperformed LCL heuristics in solution quality as well as running time. Finally, we
performed tests on LCN heuristics, varying the limit on total number of improvement over negative cycle found among three, three, five, and ten. As obvious, with relaxation in the limit, both algorithms generated better results. However, the average solution time as well as solution quality was not very good. These algorithms generate solutions with better optimality gap if the number of negative cycles in the network is not very high. However, at the same time, if the limit set on number of negative cycles is within the same range as total number of cycles present in the network, the algorithm becomes close to exact algorithm, and hence, the running time increases significantly. We can conclude from these experiments that for large size instances, LUL heuristic applied on APPull algorithms performs better than all other combination of algorithms. For the SDNCC problems, we designed the same experimental set-up and varied the controlling parameters of each heuristic in the similar fashion. The comparative behavior of all heuristic algorithms was same as discussed in SDMCC problem groups. The most effective heuristic approach was LUL implemented on APPull algorithm. These results were very close to optimality in all problems groups with label limit of 1000, obtaining optimality for three out of four instances. At the same time, running time improved from around 3.5 seconds to 0.7 seconds. It was consistently better than all other heuristic approaches.

3.6 Conclusions

In this chapter, we have introduced the subset disjoint minimum/negative cost cycle problems and discussed several exact and heuristic algorithms. The exact algorithms were inspired by pre-existing dynamic programming algorithms for resource constrained shortest path and all pair shortest path algorithms. We propose the more generalized version of these algorithms along with several strategies to reduce the network and size and thereby reduce their running time. Since, these problems are used as sub-problems of more complex problems and solved repetitively, a running time of more than a second may not be effective. We developed several heuristic approaches,
Table 3-7. Comparative analysis of heuristic SDMCC algorithms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>46-21</td>
<td>-12.90</td>
<td>10</td>
<td>3.90</td>
<td>0.91</td>
<td>4.00</td>
<td>0.50</td>
<td>2</td>
<td>7.76</td>
<td>0.11</td>
<td>7.76</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1.10</td>
<td>1.00</td>
<td>1.10</td>
<td>0.58</td>
<td>4</td>
<td>2.81</td>
<td>0.61</td>
<td>2.81</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.14</td>
<td>3.67</td>
<td>0.14</td>
<td>3.69</td>
<td>8</td>
<td>0.33</td>
<td>9.92</td>
<td>0.33</td>
<td>13.74</td>
</tr>
<tr>
<td>43-22</td>
<td>-7.23</td>
<td>10</td>
<td>2.23</td>
<td>0.94</td>
<td>2.27</td>
<td>0.51</td>
<td>2</td>
<td>4.73</td>
<td>0.11</td>
<td>4.73</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.59</td>
<td>1.02</td>
<td>0.64</td>
<td>0.61</td>
<td>4</td>
<td>2.09</td>
<td>0.62</td>
<td>2.09</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.09</td>
<td>3.57</td>
<td>0.09</td>
<td>3.56</td>
<td>8</td>
<td>0.45</td>
<td>6.37</td>
<td>0.45</td>
<td>8.24</td>
</tr>
<tr>
<td>37-17</td>
<td>-5.29</td>
<td>10</td>
<td>1.06</td>
<td>0.87</td>
<td>0.82</td>
<td>0.48</td>
<td>2</td>
<td>1.82</td>
<td>0.11</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.18</td>
<td>0.88</td>
<td>0.24</td>
<td>0.52</td>
<td>4</td>
<td>0.24</td>
<td>0.56</td>
<td>0.24</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.00</td>
<td>2.20</td>
<td>0.00</td>
<td>2.07</td>
<td>8</td>
<td>0.00</td>
<td>2.76</td>
<td>0.00</td>
<td>3.06</td>
</tr>
<tr>
<td>40-54</td>
<td>-18.22</td>
<td>10</td>
<td>5.48</td>
<td>0.93</td>
<td>5.74</td>
<td>0.50</td>
<td>2</td>
<td>8.33</td>
<td>0.11</td>
<td>8.33</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>2.33</td>
<td>1.00</td>
<td>2.65</td>
<td>0.59</td>
<td>4</td>
<td>5.06</td>
<td>0.59</td>
<td>5.06</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.39</td>
<td>3.74</td>
<td>0.41</td>
<td>3.60</td>
<td>8</td>
<td>2.02</td>
<td>8.92</td>
<td>2.02</td>
<td>12.54</td>
</tr>
</tbody>
</table>
Table 3-8. Comparative analysis of heuristic SDNCC algorithms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>46-21</td>
<td>-12.90</td>
<td>10</td>
<td>2.71</td>
<td>0.363</td>
<td>3.10</td>
<td>0.202</td>
<td>2</td>
<td>7.76</td>
<td>0.111</td>
<td>7.76</td>
<td>0.113</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>0.38</td>
<td>0.444</td>
<td>0.29</td>
<td>0.233</td>
<td>4</td>
<td>2.81</td>
<td>0.273</td>
<td>2.81</td>
<td>0.168</td>
<td>5</td>
<td>3.24</td>
<td>0.271</td>
</tr>
<tr>
<td>1000</td>
<td>0.00</td>
<td>0.651</td>
<td>0.00</td>
<td>0.463</td>
<td>8</td>
<td>0.33</td>
<td>0.558</td>
<td>0.33</td>
<td>0.423</td>
<td>10</td>
<td>0.71</td>
<td>0.593</td>
</tr>
<tr>
<td>43-22</td>
<td>-7.23</td>
<td>10</td>
<td>3.05</td>
<td>0.291</td>
<td>2.82</td>
<td>0.183</td>
<td>2</td>
<td>4.95</td>
<td>0.114</td>
<td>4.95</td>
<td>0.122</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>0.18</td>
<td>0.325</td>
<td>0.32</td>
<td>0.204</td>
<td>4</td>
<td>2.14</td>
<td>0.246</td>
<td>2.14</td>
<td>0.170</td>
<td>5</td>
<td>1.68</td>
<td>0.256</td>
</tr>
<tr>
<td>1000</td>
<td>0.05</td>
<td>0.429</td>
<td>0.05</td>
<td>0.298</td>
<td>8</td>
<td>0.45</td>
<td>0.379</td>
<td>0.45</td>
<td>0.284</td>
<td>10</td>
<td>0.27</td>
<td>0.359</td>
</tr>
<tr>
<td>37-17</td>
<td>-5.29</td>
<td>10</td>
<td>0.65</td>
<td>0.308</td>
<td>0.71</td>
<td>0.190</td>
<td>2</td>
<td>2.24</td>
<td>0.111</td>
<td>2.24</td>
<td>0.113</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>0.00</td>
<td>0.325</td>
<td>0.06</td>
<td>0.198</td>
<td>4</td>
<td>0.24</td>
<td>0.275</td>
<td>0.24</td>
<td>0.167</td>
<td>5</td>
<td>0.00</td>
<td>0.377</td>
</tr>
<tr>
<td>1000</td>
<td>0.00</td>
<td>0.334</td>
<td>0.00</td>
<td>0.193</td>
<td>8</td>
<td>0.00</td>
<td>0.336</td>
<td>0.00</td>
<td>0.195</td>
<td>10</td>
<td>0.00</td>
<td>0.368</td>
</tr>
<tr>
<td>40-54</td>
<td>-18.22</td>
<td>10</td>
<td>5.98</td>
<td>0.365</td>
<td>5.98</td>
<td>0.210</td>
<td>2</td>
<td>8.43</td>
<td>0.118</td>
<td>8.43</td>
<td>0.117</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>1.54</td>
<td>0.425</td>
<td>1.44</td>
<td>0.227</td>
<td>4</td>
<td>5.06</td>
<td>0.300</td>
<td>5.06</td>
<td>0.180</td>
<td>5</td>
<td>6.15</td>
<td>0.232</td>
</tr>
<tr>
<td>1000</td>
<td>0.00</td>
<td>0.694</td>
<td>0.00</td>
<td>0.495</td>
<td>8</td>
<td>2.02</td>
<td>0.867</td>
<td>2.02</td>
<td>0.893</td>
<td>10</td>
<td>2.44</td>
<td>0.421</td>
</tr>
</tbody>
</table>
each controlling a specific parameter of exact algorithms. Among all these heuristics, limited unprocessed labels (LUL) implemented on APPull algorithms performed better than all other heuristic combinations for SDMCC as well as SDNCC problems. These arguments were supported by extensive computational tests performed on wide variety of instances. We can also conclude that if the network contains a negative cycle, then to find the minimum cycles, algorithms specific for SDNCC problems should be used as optimal cycle must be negative.
CHAPTER 4
COLUMN GENERATION APPROACH FOR LOCATION ROUTING PROBLEMS

4.1 Introduction

The location of the facilities in the vehicle routing problem has a great effect in the total cost of the distribution system. Similarly, consideration of the possible vehicle routes while solving a facility location problem has significance in minimizing the total system cost. Therefore, it is necessary to consider vehicle routing and facility locations simultaneously. However, it is argued that the horizon of these two problems differ. The routing problems are solved for short time horizons (even daily) and they are on the tactical and operational level, but the location problems are solved for long time horizons and they are on the strategic level. In spite of this, there are many studies that give examples of practical applications that combine location and routing decisions. Blood bank location [62], newspaper distribution [41], post box location [46], waste collection [44], goods distribution [65] and parcel delivery [74] are just some of the examples. The problem that integrates the facility location and vehicle routing problems together is the combined location-routing problem (LRP) [64, 75]. The LRP consists of selecting a subset of locations among given candidate facility locations and determining the vehicle routes to visit all the customers from the selected facility locations. The objective in LRP is to minimize the sum of the fixed costs, variable costs and delivery costs such that capacities of selected facility locations and vehicles are not exceeded while the demand of each customer is satisfied. The fixed costs include the costs of establishing depot on a candidate location and the variable costs depend on the per unit throughput to a customer from the selected facility.

The LRP reflects the interdependence between location costs and routing costs [69, 75]. Hence, it is a combination of two already difficult problems: Facility Location Problem (FLP) and Vehicle Routing Problem (VRP). Both FLP and VRP have been shown to be NP-hard [23, 42, 52], thus the LRP is NP-hard as well.
The organization of the chapter is as follows. In section 4.2, we give the problem description and in section 4.3, we introduce the mixed integer linear programming formulation for the LRP. Proposed column-generation algorithm is presented in detail in section 4.4. We discuss the master problem and the pricing subproblem. We present the dynamic programming algorithm developed for the pricing subproblem. At the end of this section, we give the overall algorithm and explain the steps of the algorithm. In section 4.5, the implementation details and results are given.

4.2 Problem Description

The LRP is considered on a directed graph $G = (V, A)$, where $A$ is the set of arcs and $V = \{1, \ldots, N, N + 1, \ldots, N + M, N + M + 1, \ldots, N + 2M\}$ is the set of all nodes. The set of customers are labelled from 1 through $N$, $C = \{1, 2, \ldots, N\}$ and the set of candidate locations are labelled from 1 through $M$, $J = \{1, 2, \ldots, M\}$. The $M$ candidate facility locations are represented twice as depots. Therefore, we form two sets of locations, one is the set of departure locations, $D = \{N + 1, \ldots, N + M\}$ and second is the set of arrival locations, $A = \{N + M + 1, \ldots, N + 2M\}$. As an example, the corresponding arrival location for the departure location $N + 1$ becomes $N + M + 1$ in this representation. The set $D$ is incident to only outgoing arcs and the set $A$ is incident to only incoming arcs. Each candidate facility location $j$ has capacity $T_j$. Total demand of customers assigned to a facility $j$ cannot exceed the facility capacity $T_j$. There is a fixed cost, $FC_j$, for establishing a depot on each candidate facility location $j$ and a variable cost, $VC_j$, that depends on the per unit throughput from each candidate location $j$. Another cost component is the arc travel cost $c_{ij}$ for each arc $(i, j) \in A$. Travel cost of each arc $(i, j) \in A$ is calculated by multiplying cost per mile (CM) parameter by the Euclidean distance between nodes $i$ and $j$ ($c_{ij} = CM \ast d_{ij}$). There is an associated demand, $d_i$, with each customer. Each vehicle has a capacity $Q$ and the sum of the demands of the customers at a given route should not exceed the vehicle capacity. The number of vehicles available ($K_{UB}$) is assigned as given if it is mentioned by the corresponding benchmark case, otherwise it
is taken as 25. In order to understand the problem better, an example data for the LRP is illustrated in the Figure 4-1.

<table>
<thead>
<tr>
<th>Facility No.</th>
<th>Xcoord.</th>
<th>Ycoord.</th>
<th>Capacity (T)</th>
<th>Fixed Cost (FC)</th>
<th>Variable Cost (VC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24</td>
<td>24</td>
<td>500</td>
<td>50</td>
<td>0.02</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>21</td>
<td>500</td>
<td>50</td>
<td>0.02</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>21</td>
<td>500</td>
<td>50</td>
<td>0.03</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>24</td>
<td>500</td>
<td>50</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>22</td>
<td>500</td>
<td>50</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Vehicle Number (K) 25
Vehicle Capacity (Q) 300
Cost Per Mile (CM) 0.75

Cust No. | Xcoord. | Ycoord. | Demand |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23</td>
<td>35</td>
<td>125</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>43</td>
<td>84</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>45</td>
<td>60</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>34</td>
<td>30</td>
<td>18</td>
<td>160</td>
</tr>
<tr>
<td>35</td>
<td>10</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>36</td>
<td>34</td>
<td>60</td>
<td>80</td>
</tr>
</tbody>
</table>

Figure 4-1. Example data for the LRP

4.3 Mathematical Model

The LRP model is presented using the 2-index variables $x_{ij}$. The variable $x_{ij}$ is equal to 1 if a vehicle travels from node $i$ to node $j$ along arc $(i, j) \in A$ and 0 otherwise. The variables $x_{ij}$ are defined if sum of the demands of node $i$ and $j$ is less than vehicle capacity ($d_i + d_j \leq Q$) and if $i \neq j$. Furthermore, we do not define the variables $x_{ij}$ if $i \in V$ and $j \in D$, because no arcs can enter to the set of departure locations. Similarly we do not define $x_{ij}$ if $i \in A$ and $j \in V$, because no arcs can leave arrival locations. The $y_j (\forall j \in J)$ variables are defined for each candidate facility location $j$. The variable $y_j$ is equal to 1 if candidate facility $j$ is selected; otherwise it is 0. The $z_{ij} (\forall i \in C, j \in J)$ variable is equal to 1 if customer $i$ is assigned to the facility $j$ and 0 otherwise. The $U_i (\forall i \in C)$ variables correspond to the load on the vehicle at departure from node $i$. Their values are expected to increase along a route since the model is designed as a pick-up problem, that is, each vehicle leaves the depot empty, collects items from customers and returns to the depot with a full load. However, with a slight change on the constraint set,
the model can also be modified for service type of delivery. The LRP is formulated as a
MILP problem as follows:

Sets:
\[ J = \{1, 2, \ldots, M\} \] : set of candidate locations
\[ D = \{N + 1, N + 2, \ldots, N + M\} \] : set of departure locations
\[ A = \{N + M + 1, N + M + 2, \ldots, N + 2M\} \] : set of arrival locations
\[ C = \{1, 2, \ldots, N\} \] : set of customers

Model Parameters:
\[ N \] : Number of customers
\[ M \] : Number of candidate facility locations
\[ K \] : Number of available vehicles
\[ d_{ij} \] : Travel distance between nodes \( i \) and \( j \); \( i, j \in V \)
\[ Q \] : Vehicle capacity
\[ CM \] : Cost per mile
\[ d_i \] : Demand of customer \( i \); \( i \in C \)
\[ T_j \] : Capacity of candidate facility \( j \); \( j \in J \)
\[ FC_j \] : Fixed cost of candidate facility \( j \); \( j \in J \)
\[ VC_j \] : Variable cost of candidate facility \( j \); \( j \in J \)

Decision Variables:
\[ x_{ij} \] : 1 if the vehicle goes from node \( i \) to node \( j \), 0 otherwise; \( i \in C \cup D, j \in C \cup A \)
\[ y_j \] : 1 if the candidate facility \( j \) is selected, 0 otherwise; \( j \in J \)
\[ z_{ij} \] : 1 if customer \( i \) is assigned to facility \( j \), 0 otherwise; \( i \in C, j \in J \)
\[ U_i \] : Load at departure from customer \( i \); \( i \in C \)
Minimize

\[
\sum_{j \in J} FC_j y_j + \sum_{j \in J} \sum_{i \in C} VC_j d_{ij} z_{ij} + \sum_{i \in C \cup D} \sum_{j \in C \cup A} c_{ij} x_{ij}
\]  

(4–1)

subject to

\[
\sum_{j \in C \cup A} x_{ij} = 1, \quad \forall i \in C,
\]  

(4–2)

\[
\sum_{j \in C \cup D} x_{ji} = 1, \quad \forall i \in C,
\]  

(4–3)

\[
\sum_{i \in C} x_{ji} = \sum_{i \in C} x_{i(j+M)}, \quad \forall j \in D,
\]  

(4–4)

\[
K_{LB} \leq \sum_{j \in D} \sum_{i \in C} x_{ji} \leq K_{UB},
\]  

(4–5)

\[
U_i - U_j + Q x_{ij} \leq Q - d_j, \quad \forall i, j \in C,
\]  

(4–6)

\[
d_i \leq U_i \leq Q, \quad \forall i \in C,
\]  

(4–7)

\[
\sum_{i \in C} d_{ij} z_{ij} \leq T_j y_j, \quad \forall j \in J,
\]  

(4–8)

\[
z_{ij} \leq y_j, \quad \forall i \in C, \forall j \in J,
\]  

(4–9)

\[
\sum_{j \in J} z_{ij} = 1, \quad \forall i \in C,
\]  

(4–10)

\[
x_{i(j+N+M)} \leq z_{ij}, \quad \forall i \in C, \forall j \in J,
\]  

(4–11)

\[
x_{i(j+N+M)} \leq z_{ij}, \quad \forall i \in C, \forall j \in J,
\]  

(4–12)

\[
x_{ij} + z_{ik} - z_{jk} \leq 1, \quad \forall i, j \in C, \forall k \in J,
\]  

(4–13)

\[
x_{ij} + z_{jk} - z_{ik} \leq 1, \quad \forall i, j \in C, \forall k \in J,
\]  

(4–14)

\[
x_{ij} \in \{0, 1\}, \quad \forall i \in C \cup D, \forall j \in C \cup A \mid i \neq j
\]  

(4–15)

\[
z_{ik} \in \{0, 1\}, \quad \forall i \in C, \forall k \in J
\]  

(4–16)

\[
y_{j} \in \{0, 1\}, \quad \forall j \in J
\]  

(4–17)

The objective function is to minimize the total cost which is composed of the sum of fixed costs of establishing depots, the sum of variable costs that depend on each depot’s throughput and the sum of delivery costs. Constraints (4–2) state that there is
exactly one arc leaving each customer. Similarly, constraints (4–3) state that there is exactly one arc coming into each customer. Constraints (4–4) ensure that the number of vehicles leaving a departure depot equals to the number of vehicles returning to its corresponding arrival depot. Constraint (4–5) enforces that the total number of vehicles used should not exceed the upper bound on the number of vehicles and it should be greater than \( K_{LB} \) which is calculated as \( \lceil \sum_{i \in C} d_i / Q \rceil \). In constraint set (4–6), the valid inequality that was proposed by Kulkarni and Bhave [45] is used in order to eliminate cycles that can occur between customers. Constraints (4–7) give the lower and upper bound of the \( U_i \) variables. The load on vehicle at departure from a node \( i \) can be at least equal to the demand of that node and it can at most be equal to the capacity of the vehicle. Constraints (4–8) ensure that no demand is assigned to a location if that location is not selected (that is, \( y_j = 0 \)) and if a candidate location is selected (that is, \( y_j = 1 \)) then the sum of the demands that are assigned to the selected facility should not exceed the capacity of the facility that will be constructed on that location. Constraints (4–9) specify that customer \( i \) can be assigned to a location \( j \) only if that location is selected. Constraints (4–10) require that each customer is assigned to exactly one depot. Constraints (4–11) state that if a customer is not assigned to a location then there cannot be an arc that goes from that customer to that location’s arrival node. Similarly, constraints (4–12) state that if a customer is not assigned to a location then there cannot be an arc that goes from the departure node of that location to this customer. Constraints (4–13) and (4–14) ensure that if customer \( j \) follows customer \( i \) and if customer \( i \) is assigned to candidate location \( k \) then customer \( j \) must also be allocated on that same candidate location. The remaining constraints (4–15),(4–16) and (4–17) specify that the routing variables \( (x_{ij}) \), the allocation variables \( (z_{ik}) \) and the location variables \( (y_j) \) can take values 0 or 1.

The given exact model is observed to solve small size LRPCs (up to 15 customers and 4 depots) optimally in the computational experiments that we conducted. Hence, we
propose a column generation algorithm tailored to the solution of the LRP. The algorithm is presented in section 4.4 in detail.

4.4 Column Generation Algorithm

4.4.1 The Master Problem

A feasible solution of a LRP is composed of the selected subset of locations and a set of feasible routes outgoing from these selected locations. A route is declared as feasible if it does not violate vehicle capacity restrictions, if each customer assigned to the route is visited exactly once (it means that the route should be an elementary route) and if the route starts and ends at the same depot. The set of feasible routes that can exist in the optimal solution of LRP is considered while constructing the LRP master problem. Initially, if we assume that we have all feasible routes then the LRP reduces to selecting a subset of these feasible columns such that the cost of the routes and fixed cost of opening the facilities that these routes originate are minimized. Each feasible route forms a binary variable (that is, a column) of the MP model. The MP model solves the LRP if all the feasible columns, the set R, is readily available. Given the set R, the task of the MP model reduces to assign all customers to the routes such that each customer is assigned to exactly one route and MP also handles the facility capacity constraints so that feasibility is not violated. That is why, the MP is a set partitioning problem with side constraints. The natural issues that come to mind are the difficulty of finding the feasible routes efficiently and the exponential number of feasible routes that should be considered. The problem of huge number of feasible routes is addressed by relaxing the master problem which is an integer programming problem and solving the resultant linear program (RMP) by column generation. We start the column generation algorithm by considering only a subset of feasible columns. Then using the insight of solving a linear programming (LP) problem, we add only the columns that have negative reduced costs to the master problem. The problem of finding the feasible routes that have negative reduced costs is solved in the subproblem of the column generation
algorithm and explained in the section 4.4.2. The mathematical formulation of the master problem is given below:

**Sets:**

- \( R \): set of feasible routes
- \( J = \{1, 2, \ldots, M\} \): set of candidate locations
- \( C = \{1, 2, \ldots, N\} \): set of customers

**Model Parameters:**

- \( K \): Number of available vehicles
- \( \delta_{ir} \): 1 if the customer \( i \) is assigned to route \( r \), 0 otherwise; \( i \in C, r \in R \)
- \( \mu_{jr} \): 1 if the route \( r \) originates from facility \( j \), 0 otherwise; \( j \in J, r \in R \)
- \( PD_r \): Demand of route \( r \); \( r \in R \)
- \( C_r \): Cost of route \( r \); \( r \in R \)
- \( T_j \): Capacity of candidate facility \( j \); \( j \in J \)
- \( FC_j \): Fixed cost of candidate facility \( j \); \( j \in J \)

**Decision Variables:**

- \( x_r \): 1 if the route \( r \) is selected, 0 otherwise; \( r \in R \)
- \( y_j \): 1 if the candidate facility \( j \) is selected, 0 otherwise; \( j \in J \)

Minimize

\[
\sum_{r \in R} C_r x_r + \sum_{j \in J} FC_j y_j
\]  

subject to

\[
\sum_{r \in R} \delta_{ir} x_r = 1, \quad \forall i \in C,
\]

\[
y_j \leq \sum_{r \in R} \mu_{jr} x_r \leq K y_j, \quad \forall j \in J,
\]

\[
\sum_{r \in R} \delta_{ir} \mu_{jr} x_r \leq y_j, \quad \forall i \in C, \quad \forall j \in J,
\]

\[
\sum_{j \in J} y_j \geq LB_w,
\]

\[
LB_v \leq \sum_{r \in R} x_r \leq K,
\]
The objective function of the MP is to minimize the cost of the selected routes among the set of feasible routes $R$ and the fixed cost of selected facility locations. Objective function values of the MP correspond to the exact LRP model presented previously. Note that the variable costs for each selected location is incorporated to the cost of each route found. Hence, the cost of a new route $r = \{i_1, i_2, \ldots, i_t\}$ can be found by the equation (4–27):

$$C_r = CM \sum_{s=1}^{s=t-1} d_{i_s i_{s+1}} + VC_{i_1} \sum_{s=2}^{s=t-1} q_s$$  \hspace{1cm} (4–27)$$

Note that in route $r$, $i_1$ is the departure node of a candidate location and $i_t$ is the corresponding arrival node of this candidate location. We add the demand of the customer nodes $\{i_2, \ldots, i_{t-1}\}$ only in the second term of the formula and obtain the route demand and multiply it by the variable cost of the candidate location $i_1$. Remember that variable cost is the cost per unit throughput from the given location.

Constraints (4–19) are defined for each customer and form the set partitioning constraints. These constraints state that each customer $i$ is served by exactly one route among the set of feasible routes $R$. Constraint set (4–20) restricts the number of paths originating from a candidate facility location. If a candidate location $j$ is selected then it requires that at least one route originates from that location and the maximum number of routes that can originate from the location is set to the available number of vehicles. If the candidate location $j$ is not selected then it requires that no routes will originate from this location. Constraint set (4–21) is first given in [14]. One classical constraint that can be written instead of this constraint could be $x_r \leq \sum_{j \in J} \mu_j y_j$, $\forall r \in R$. The classical

$$\sum_{r \in R} PD_r \mu_{jr} x_r \leq T_j y_j, \quad \forall j \in J,$$  \hspace{1cm} (4–24)$$

$$x_r \in \{0, 1\}, \quad \forall r \in R$$  \hspace{1cm} (4–25)$$

$$y_j \in \{0, 1\}, \quad \forall j \in J$$  \hspace{1cm} (4–26)$$

97
one is written for each route \( r \) and simply states that if a route \( r \) that starts at facility \( j \) is selected, then facility \( j \) must also be selected. Note that the right term corresponds to the facility from which route \( r \) originates. For the preferred constraint set let \( R_{ij} \) denote the set of routes that includes customer \( i \) and depot \( j \) pair simultaneously. It is clear that the sum of route selection variables over the set \( R_{ij} \) must be at most one since only one route that includes the same \( i - j \) pair can be selected in a feasible solution. So this can be written by the following constraint:

\[
\sum_{r \in R_{ij}} x_r \leq y_j, \quad \forall i \in C, \: \forall j \in J
\]  

(4–28)

Note that the left side of the constraint (4–28) is tighter when compared to the classical one. Therefore, relaxation values of the model with constraint set (4–28) will be greater than or equal to relaxation values with the classical model. Another advantage is the reduced number of constraints since (4–28) is defined for each customer and candidate location. However, the classical constraint is defined for each feasible route \( R \). Note that by using the \( \delta_{ir} \) and \( \mu_{jr} \) parameters, (4–28) can be rephrased as the constraint (4–21), that is, they are equivalent.

Constraint (4–22) sets the lower bound on the number of facilities that can be selected among the candidate set. \( LB_w \) is calculated as \( \lceil \sum_{i \in C} d_i / T_{max} \rceil \) where \( T_{max} \) is the maximum candidate facility capacity among all. Constraint (4–23) states that the total number of routes selected (or vehicles used) should not exceed the the number of available vehicles and it should be greater than \( LB_v \) which is calculated as \( \lceil \sum_{i \in C} d_i / Q \rceil \). Constraint set (4–24) requires that sum of the demands of the customers assigned to a facility should not exceed the facility’s capacity if the facility is selected. The constraints (4–25) and (4–26) specify that the route selection variables \( (x_r) \) and the location variables \( (y_j) \) can take values 0 or 1.
4.4.2 The Pricing Problem

The promising feasible columns are generated in the subproblem of the column generation and added to the master problem. In a simplex algorithm for a minimization problem, a promising nonbasic variable is the one with the negative reduced cost. The reduced cost defines the reduction in the objective function if the nonbasic variable is increased by one unit. Recall from the simplex algorithm that the reduced cost of a variable $x_r$ is given by $\hat{c}_r = c_r - c_B B^{-1} A_r$. In this notation $A_r$ is the $r^{th}$ column corresponding to the variable $x_r$ and $c_r$ is the objective function coefficient of the variable $x_r$. Note that $c_B B^{-1}$ gives the dual values ($\pi$) of the solution of the LP model. So given the dual values obtained from the solution of the relaxed master problem (RMP), the objective of the subproblem is to find a new feasible column such that reduced cost of the column is negative, that is the column is promising. The new columns identified in the subproblem are added to the set R, prepared for the RMP and then RMP model is resolved. Column generation algorithm is stopped when the pricing subproblem cannot identify a feasible route with a negative reduced cost.

We can identify promising columns by solving an elementary shortest path problem with resource constraints (ESPPRC) for each candidate location $j$. The subproblem is considered on a directed subgraph $G^j = (V^j, A^j)$, where $A^j$ is the set of arcs and $V^j = \{0, 1, ..., N, N + 1\}$ is the set of nodes. For the candidate location $j$, the departure and arrival nodes are shown by the nodes 0 and $N + 1$, respectively. A typical route will start from node 0 and end with node $N + 1$. An important point is the cost of paths in the subproblem for the network constructed for a facility location $j$; the cost of any path found must correspond to the reduced cost of the path variable in the master problem. In other words, the sum of the arc costs of a route identified in a subproblem must give the reduced cost of that route if it had been in the MP model with the current dual values. Table 4-1 shows the constraint number and associated dual values that are obtained as the master problem is solved. Thus, the modified arc cost $\hat{c}_{ij}$ in the constructed network

99
Table 4-1. Constraints and associated dual values for the Master Problem

<table>
<thead>
<tr>
<th>MP constraint number</th>
<th>Dual value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4–19</td>
<td>$\pi^{4-19}_i \forall i \in C$</td>
</tr>
<tr>
<td>4–20</td>
<td>$\pi^{4-20}_j \forall j \in J$</td>
</tr>
<tr>
<td>4–21</td>
<td>$\pi^{4-21}_i \forall i \in C$</td>
</tr>
<tr>
<td>4–23</td>
<td>$\pi^{4-23}_i$</td>
</tr>
<tr>
<td>4–24</td>
<td>$\pi^{4-24}_j \forall j \in J$</td>
</tr>
</tbody>
</table>

of the subproblem is given by;

$$
\tilde{c}_{ij} = \begin{cases} 
    c_{ij} - \pi^{4-19}_i + \pi^{4-21}_i - \pi^{4-24}_0 \ast q[i] & \text{if } i \in C \text{ and } j \in C \cup N+1 \\
    c_{ij} - \pi^{4-20}_i - \pi^{4-23}_i & \text{if } i = 0 \text{ and } j \in C
\end{cases}
$$

where

$$
    c_{ij} = \begin{cases} 
    CM \ast d_{ij} + q_i \ast VC_0 & \text{if } i \in C \text{ and } j \in C \cup N+1 \\
    CM \ast d_{ij} & \text{if } i = 0 \text{ and } j \in C
\end{cases}
$$

Note that modified arc costs are calculated only if the corresponding arc variables are defined. If arc variables are undefined, we set the arc costs to infinity indicating the absence of arc.

Based on our observations in the computational experiments, one of the most critical issues to create a time-efficient algorithm is to solve the subproblem effectively. The subproblem which is an ESPPRC was shown to be strongly NP-Hard by Dror [29]. An integer model whose solution identifies an elementary shortest path on the defined subnetwork for a location $j$ would give only the column with the most negative reduced cost. However, the goal is to determine whether a negative reduced cost column exists or not. So obtaining a set of good routes for each location $j$ is more important than finding the best route for a fast evolving algorithm. That is why, for the pricing subproblem we use dynamic programming to find a set of routes that have negative reduced costs. The structure of the algorithm we used for solving the ESPPRC is presented in Figure 4-2. A path is called non-elementary if a node is visited more than once hence if the path contains any loop; otherwise it is called an elementary path. The
dynamic programming algorithm proposed solves elementary shortest path problem with resource constraints by modifying the algorithm of Larsen [51] that solves the relaxed version of the ESPPRC in which non-elementary paths are allowed. In his dynamic programming algorithm, Larsen [51] uses the basic principles of Dijkstra’s algorithm in order to find feasible routes with negative reduced costs. Normally, an optimal solution to a LRP does not involve cycles since each customer is visited once. However, cycle constraint is generally relaxed and a shortest path problem with resource constraints (SPPRC) is solved. SPPRC is also NP-Hard but pseudo-polynomial algorithms exist to solve the SPPRC [27] hence they can be solved more efficiently (e.g. see [5]). Using this solution approach, the solution of the RMP might contain non-elementary paths since non-elementary paths are added to the MP. However, it has been shown that an optimal integer solution will only contain elementary paths because of the triangular inequality assumption [20]. The disadvantage of this approach is that the lower bound obtained by the solution of RMP is weakened since negative cycles help to reduce costs. In our dynamic programming algorithm we store the predecessor nodes for each partial path and eliminate k-cycles.

The modified arc costs are updated each time RMP is solved. We create the initial label \( \{0,0\} \) in step 1 of the dynamic programming algorithm. Each label \( \{i, \text{AccDem}_i\} \) represents a partial path. The first entry in a label is the last node of the partial path and the second is the accumulated demand for the partial path. Cost of a label is shown by \( \text{Cost}(\{i, \text{AccDem}_i\}) \). The cost of the initial label is set to 0. When a label is created it is added to the set of unprocessed labels. In step 2, \textit{BestLabel()} function chooses the label that has minimum accumulated demand among the unprocessed labels. Since the labels are processed in order of increasing accumulated demand, a label can be chosen and processed only one time in step 2. Note that the label is removed from the set of unprocessed labels at the end of step 2 after it has been processed. The label \( \{u, \text{AccDem}_u\} \) that has the minimum accumulated demand is extended to the other
nodes if the extension is feasible and the cost is improved. If a label could be created after the feasibility and improvement checks, it is added to the set of unprocessed labels. 

CreateLabel() function checks whether the label \(\{v, AccDem_v\}\) that will be created as the result of extension already exists or not. If it exists the cost of old label is updated, otherwise the label is new and its cost is taken as infinite while doing the improvement check.

Dynamic Programming Procedure for solving ESPPRC

Inputs: Updated arc costs, demand of each node and vehicle capacity
Output: A set of elementary paths from node 0 to node \(N + 1\) that have negative reduced costs and do not violate any capacity constraints.

Step 1. Initialization
- Get the updated arc costs associated with the last RMP solved
- CreateLabel(\{0, 0\})
- Initialize a set of sets that keep elements of each distinct path found
- Initialize predecessor set

Step 2. WHILE \(u \neq N + 1\) DO
- \(\{u, AccDem_u\} = \text{BestLabel(unprocessed labels)}\)
- IF \(u < N + 1\) THEN
  - FOR each \(v \in C \cup \{N + 1\}\) DO
    - Feasibility check:
      - IF \(u \neq v\) AND \(\hat{c}_{uv} \neq \infty\) AND AccDem\(_u + d_v <= Q\) & \(v \notin \text{preds}[\{u, AccDem_u\}]\) THEN
        - AccDem\(_v = AccDem_u + d_v\)
    - Improvement check:
      - IF Cost(\(\{u, AccDem_u\}\) + \(\hat{c}_{uv}\) < Cost(\(\{v, AccDem_v\}\)) THEN
        - isCreateLabel = DominanceRule()
      - IF isCreateLabel = true THEN
        - CreateLabel(\(\{v, AccDem_v\}\))
        - \(\text{preds}[\{v, AccDem_v\}] = \text{preds}[\{u, AccDem_u\}] \cup \{v\}\)
      - IF \(v = N + 1\) AND Cost(\(\{u, AccDem_u\}\) + \(\hat{c}_{uv}\) < 0 THEN
        - Form a set from the nodes of the distinct path
        - IF The node set of the path is distinct THEN
          - Keep the order of the node set
        - ELSE IF The node set already exists THEN
          - Update the order of the node set
        - Remove label \(\{u, AccDem_u\}\) from the set of unprocessed labels
  - END FOR
- END IF
- IF \(v = N + 1\) AND Cost(\(\{u, AccDem_u\}\) + \(\hat{c}_{uv}\) < 0 THEN
  - Form a set from the nodes of the distinct path
  - IF The node set of the path is distinct THEN
    - Keep the order of the node set
  - ELSE IF The node set already exists THEN
    - Update the order of the node set
  - Remove label \(\{u, AccDem_u\}\) from the set of unprocessed labels
- END IF

Step 3. FOR each distinct path sets DO
- Prepare the routes found for the MP

Figure 4-2. Dynamic programming algorithm for ESPPRC
In order to prevent visits to nodes more than once, we keep the predecessor nodes of each label in \( \text{CreateLabel}() \) function and during the feasibility check we extend a label \( \{u, \text{AccDem}_u\} \) to a node \( v \) only if node \( v \) is not an element of the partial path ending at node \( u \). The improvement check is positive if the cost of the old label can be decreased or if the label is created for the first time. However, we do not create all the labels that can pass the improvement check. The reason is that there may be exponentially many partial paths that are feasible even for small size problems and in order to have a more efficient algorithm we may not create labels that are dominated by other labels. The basic dominance used for solving the non-elementary shortest path problem with resource constraints is as follows: Let \( \{i, \text{AccDem}_1\} \) and \( \{i, \text{AccDem}_2\} \) be two partial paths ending at the same node \( i \). Label \( \{i, \text{AccDem}_1\} \) dominates label \( \{i, \text{AccDem}_2\} \) only if the following conditions are satisfied:

\[
\text{Cost}(\{i, \text{AccDem}_1\}) < \text{Cost}(\{i, \text{AccDem}_2\}) \quad (4–29)
\]
\[
\text{AccDem}_1 \leq \text{AccDem}_2 \quad (4–30)
\]

If an extension of the second path is feasible then the same extension will also be feasible for the first path with a lower cost considering the conditions given in Eqs. (4–29) and (4–30). Thus, the label for the second path can be discarded. However, for the elementary shortest path problem with resource constraints the basic dominance rule cannot be applied directly and a modified dominance rule has been proposed first by Dumas and Desrosier [30] and Chabrier [20]. For SPPRC, the first and second path can both continue on the same node if vehicle capacity is not exceeded. However, for ESPPRC the second path has the chance of continuing with a node of the first path and this extension might be useful later. Note that the first path cannot continue with a node already present in its partial path. Hence, the promising node cannot be added to the first path. A third condition that is required for the first path to dominate the second is
given in (4–31).
\[ S_{\{i,AccDem_1\}} \subseteq S_{\{i,AccDem_2\}} \quad (4–31) \]

\( S_{\{i,AccDem_1\}} \) denotes the set of nodes of the first partial path. The third condition requires that the set of nodes of the first path is included in the second path so that possible successors of both paths are the same. Similar to [20], our computational experience also shows that if all the three conditions are applied, the number of labels discarded due to dominance are much less compared to basic dominance. Hence, our strategy is to apply basic dominance as long as the algorithm could find columns with negative costs. When the algorithm cannot identify a column with negative cost, the third condition is also applied to decide on dominance. As the algorithm continues, if a label \( \{N + 1, AccDem_v\} \) with negative reduced cost is discovered, the set \( S_{\{N+1,AccDem_v\}} \) is formed from the nodes of the label. Let \( S_{N+1} \) denotes the set that keeps sets of each distinct path as its elements. If \( S_{\{N+1,AccDem_v\}} \in S_{N+1} \), then the path is already discovered and we only update the order of the nodes. But if \( S_{\{N+1,AccDem_v\}} \notin S_{N+1} \), then this set is a newly discovered distinct path and we add the path to the set \( S_{N+1} \). The solution of the subproblem for facility location \( j \) is terminated when the unprocessed labels are only the ones associated with \( N + 1 \). The labels \( \{N + 1, AccDem_v\} \) are not processed since node \( N + 1 \) is the arrival node and hence they cannot be extended to a further node.

If \text{BestLabel}() \ function must return a label of node \( N + 1 \) since all the other labels are processed then the algorithm continues with step 3. In this step, all the distinct paths associated with node \( N + 1 \) are prepared for and added to the master problem.

### 4.4.3 The Algorithm

Column generation is an effective method to solve large-scale linear programming problems. Our work proposes a column generation algorithm for the solution of capacitated location-routing problem (The vehicles and facilities are capacitated). The master problem (MP) in our column generation algorithm is a set-partition based formulation which proved to be useful for a variety of routing problems such as the
vehicle routing problem with time windows. The MP model is an integer programming
model and is explained in detail in the section 4.4.1. Column generation technique is
particularly applied for the solution of the relaxed master problem (RMP) which happens
to have an exponential number of feasible columns. The RMP is the LP relaxation of the
MP. As in any column generation technique, we start with a subset of feasible columns
and we try to find new promising columns in the subproblem which is also called the
pricing problem. For the pricing problem we solve elementary shortest path problem with
resource constraints by modifying the algorithm of Larsen [51]. The objective function
value of the RMP forms a lower bound to the original problem when no more feasible
routes with negative reduced cost can be found by the subproblem. The outline of our
column generation algorithm is given in Figure 4-3.

In the initialization step, we first import the data for the given problem. After
determining the number of customers (N) and the number of candidate locations
(M), the set of departure locations (D) and the set of arrival locations (A) are established
(Step 1.2). The arc travel costs are calculated by multiplying the Euclidean distances
between pairs of coordinates with the cost per mile parameter (CM). The distances
are calculated with five decimal point and truncation with the formula given in equation
(4–32).

\[
d_{ij} = \frac{\lfloor (10^5) \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \rfloor}{10^5}
\]  

(4–32)

The column generation algorithm starts with an initial set of feasible columns
which form a feasible solution. We could use the single-node routes, that is, we could
assign each customer to each depot and have \( N \times M \) starting routes or columns.
Note that each feasible route will be a variable or equivalently a column of the master
problem. However, this approach has some disadvantages. First, since we would have
one customer in each route, we would have \( N \) routes for the initial feasible solution.
Therefore the number of available vehicles (K) that restricts the number of routes to be
Steps of Column Generation Procedure

*Inputs:* Coordinates and demand of each node,
Variable cost, fixed cost and capacity for each candidate location
Available number of vehicles, vehicle capacity and cost per mile

*Output:* An integer solution and a gap for the given LRP

**Step 1. Initialization**
1.1 Get the inputs
1.2 Create departure depot(D) and arrival depot(A) sets
1.3 Calculate distance matrix
1.4 Initialize routes
   - Construct single-node routes
   - Construct multi-node routes using heuristics
1.5 Prepare the initial set of routes for the Master Problem (MP)

**Step 2. Construct Master Problem**
Solve Integer Master Problem with the initial set of routes

**Step 3. WHILE** $RD_{cost_r} < 0$ for any new path $r$ **DO**
3.1 Solve Relaxed Master Problem (RMP)
   - Update the lower bound
   - Get the dual values
   - Modify the edge costs using the dual values
   - **IF** the solution is not fractional
   - **OR** gap $\geq 3\%$ (checked at regular intervals) **THEN**
     - Solve Integer Master Problem
     - Update the upper bound
     - Keep the solution as the starting feasible solution

3.2 **FOR** each $j \in J$ **DO**
   - Solve Subproblem with modified costs
   - Prepare the paths with negative reduced cost for the MP

Figure 4-3. Column generation algorithm

used could not be incorporated in the master problem model in this case. Constraints (4–20) and (4–23) of the MP model must be removed to have a feasible solution and the relaxation gap increases if they are not utilized; therefore, the integer MP is solved in a longer computing time. Second, the nature of the LRP requires to fill the vehicle capacity as much as possible to force smaller number of depots and vehicles. That is why, multi-node routes that utilize the vehicle capacity better are likely to be present in optimal solutions. As expected, using single-node routes only gives worse initial feasible solutions in the sense that they are far from optimal. Hence, the number of pricing
problems that should be solved to find good quality routes increases. To construct an initial set of multi-node routes, we apply a heuristic method for each candidate depot \( j \). The routes are found by using Clarke and Wright Savings algorithm \([22]\) as a construction heuristic and it is followed by a series of route improvement heuristics, namely 2-opt, 1-1 exchange, crossover and simulated annealing.

In step 1.5, the initial set of feasible routes are prepared for the master problem. Note that each feasible route \( r \) is incorporated to the master problem as a variable \( x_r \), that is why, in order to prepare a route for the MP, the modelling components related to the variable \( x_r \) must be constructed. Hence, in the preparation step, we calculate the cost of the route \( (C_r) \), the demand of the route \( (PD_r) \) and arrange the parameter \( \delta_{ir} \) which holds the customers assigned to route \( r \) and parameter \( \mu_{jr} \) which holds the depot that is used in route \( r \). The parameter \( \delta_{ir} \) is 1 if the customer \( i \) is assigned to route \( r \), 0 otherwise and the parameter \( \mu_{jr} \) is 1 if the route \( r \) originates from depot \( j \), 0 otherwise. In step 2, we construct the MP model which is an integer programming problem and when we solve the master problem for the first time with these prepared columns, we obtain an initial feasible solution and keep the initial solution to serve as a starting integer solution for our instance. Note that by using CPLEX 11.1, MP model is constructed columnwise only one time and later the newly found columns are added to the model without reconstructing the entire model but by using the `addColumn()` routine of the CPLEX.

We iterate step 3 until we cannot find a feasible column that has negative reduced cost for any candidate location \( j \). After the relaxed master problem (RMP) is solved (step 3.1), the lower bound is updated and the edge costs are modified using the dual values. At this moment, it is very beneficial to check whether the solution of the RMP is integral or not. Computational experience showed that it is not uncommon to have integral solutions for the RMP since the size of the RMP increases step by step at each iteration of step 3.2. Especially for the initial iterations in which small size LPs are
solved, feasible region is defined well for the given MP. Therefore, it is more difficult for a fractional solution to be found. If an integral solution is found for the RMP, this integral solution is clearly the optimum of the integer MP problem for the feasible set of columns $R$ at that MP instance. This means that solving the integer problem with the given columns at hand would finish at node 0 of the branch-and-bound algorithm and hence it would be very easy to update the upper bound if the LP solution is integral. Frequent updates of the upper bound and keeping the last upper bound found as the starting initial solution for the integer MP is one of the important strategies applied in our algorithm. The goal is to keep the gap between the upper bound and the lower bound as small as possible during the execution of the algorithm. If the only upper bound kept would be the one we get at step 2 from the solution of the first integer master problem with the initial set of routes, then closing the gap when the loop at step 3 ends would be more difficult. Note that column generation is used to solve the RMP. So the lower bound at hand at the end of the algorithm would be the first relaxation of the integer MP problem while applying the branch-and-bound. It is clear that the smaller the gap is at node 0 of the branch-and-bound tree, the faster the branch-and-bound execution is going to be. To keep the gap manageable, we solve an integer problem after each 20 RMP solutions if the gap calculated at that instance is greater than 3%. In step 3.2, we solve the subproblem for each candidate location $j$ and identify the columns that will be added to the master problem and hence used in the solution of step 3.1. We restrict the maximum number of columns to be added for a single candidate location to 50 columns. Hence a maximum of $50 \times M$ columns are added in each solution of step 3.2. The algorithm continues until no more feasible routes with negative reduced cost are discovered at step 3 and the objective value of the RMP at this stage specifies the lower bound for the integer problem.
4.5 Implementation Details

4.5.1 Test Problems

We try to solve a general LRP where we have capacitated multiple facilities and multiple vehicles. There are no commonly accepted and widely studied benchmark problems for this type of LRP. The 3 benchmark problems of Perl [64] are the only problem sets that match with the general LRP we study. LRP benchmarks of Perl [64] are studied in [65], [37] and [76]. The authors of these papers compare their computational results with each other. Another LRP data set which is close to the general LRP we study is provided by Or [61] and is studied in [62]. However, facilities are uncapacitated in the LRP benchmark of Or [61]. Recently, Barreto [11] tried to form a standardized database for the general LRPs by respecting the format proposed by Perl [64]. The instances that are compiled by Barreto [11] are mostly from the literature of vehicle routing problems. The seed papers or theses for these instances are due to Perl [64], Daskin [26], Gaskell [34], Min [56], Or [61], Srivastava [70] and Christofides [21]. In order to form a standardized database Barreto took the depot variable costs as zero and take the cost per mile parameter as 1 in all the instances. However, while making computational tests we respected the original data of Perl’s 3 benchmarks. Therefore, we could make comparisons with the solutions of [65], [37] and [76]. For the other instances, we could compare our results only with [12].

4.5.2 Computational Platform

Computational experiments are performed on a 2.39 GHz PC with 1.99 GB of RAM. We report the total elapsed times, not the CPU time for each instance. We have used ILOG CPLEX version 11.1 [40]. Mixed integer and primal simplex optimizers of CPLEX are used to solve MIP and LP models respectively. The algorithm is implemented with Java API of ILOG CPLEX Concert Technology.
4.5.3 Computational Results

Performance of the column generation algorithm has been tested on the 18 benchmark instances. Names of the instances are formed by the name of the researcher, number of customers and number of candidate facility locations. For instance, Perl-55c-15d refers to the Perl’s benchmark problem with 55 customers and 15 candidate facility locations. The results obtained for the instances are presented in Table 4-2. In this detailed table, vehicle capacity (Vcap), integer objective function value (IP), lower bound obtained from the column generation procedure (LB), integrality gap (Gap), number of vehicles/routes used (Vno) and the set of locations selected (Depots) in the solution and finally the runtime of the column generation algorithm (Time(s)) is given respectively.

We reported our running times in Table 4-2. We solved 11 out of 18 instances within 1 hour and 13 instances within 4 hours. The generation of promising columns continued until an elapsed time limit of 12 hours for 5 of the instances (Christofides-100c-10d, Min-134c-8d, Daskin-88c-8d, Daskin-150c-10d, Or-117c-14d). For these instances, the final integer solution is reported and TILIM is written in the runtime column to indicate that algorithm has been terminated due to time limit.

Table 4-2. Results on the LRP benchmark instances

<table>
<thead>
<tr>
<th>No</th>
<th>Problem Name</th>
<th>Vcap</th>
<th>IP</th>
<th>LB</th>
<th>Gap (%)</th>
<th>Vno</th>
<th>Depots</th>
<th>Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gaskell-21c-5d</td>
<td>6000</td>
<td>424.899</td>
<td>424.899</td>
<td>0.00</td>
<td>4</td>
<td>1,2</td>
<td>13.39</td>
</tr>
<tr>
<td>2</td>
<td>Gaskell-22c-5d</td>
<td>4500</td>
<td>585.108</td>
<td>585.108</td>
<td>0.00</td>
<td>3</td>
<td>1</td>
<td>128.25</td>
</tr>
<tr>
<td>3</td>
<td>Gaskell-29c-5d</td>
<td>4500</td>
<td>512.103</td>
<td>504.965</td>
<td>1.39</td>
<td>4</td>
<td>2,3</td>
<td>468.80</td>
</tr>
<tr>
<td>4</td>
<td>Gaskell-32c-5d</td>
<td>8000</td>
<td>568.836</td>
<td>544.316</td>
<td>4.31</td>
<td>4</td>
<td>3</td>
<td>1233.00</td>
</tr>
<tr>
<td>5</td>
<td>Gaskell-32c-5d</td>
<td>11000</td>
<td>504.329</td>
<td>502.396</td>
<td>0.38</td>
<td>3</td>
<td>3</td>
<td>2576.16</td>
</tr>
<tr>
<td>6</td>
<td>Gaskell-36c-5d</td>
<td>250</td>
<td>460.374</td>
<td>456.257</td>
<td>0.89</td>
<td>4</td>
<td>5</td>
<td>180.98</td>
</tr>
<tr>
<td>7</td>
<td>Christofides-50c-5d</td>
<td>160</td>
<td>565.618</td>
<td>562.621</td>
<td>0.59</td>
<td>5</td>
<td>2,5</td>
<td>3235.09</td>
</tr>
<tr>
<td>8</td>
<td>Christofides-75c-10d</td>
<td>140</td>
<td>867.326</td>
<td>835.855</td>
<td>3.63</td>
<td>12</td>
<td>1,5,9,11</td>
<td>11027.27</td>
</tr>
<tr>
<td>9</td>
<td>Christofides-100c-10d</td>
<td>200</td>
<td>849.468</td>
<td>-</td>
<td>-</td>
<td>9</td>
<td>-</td>
<td>TILIM</td>
</tr>
<tr>
<td>10</td>
<td>Perl-12c-2d</td>
<td>140</td>
<td>355.582</td>
<td>355.582</td>
<td>0.00</td>
<td>2</td>
<td>1</td>
<td>1.16</td>
</tr>
<tr>
<td>11</td>
<td>Perl-55c-15d</td>
<td>120</td>
<td>5465.510</td>
<td>5400.751</td>
<td>1.18</td>
<td>10</td>
<td>2,8,12</td>
<td>284.61</td>
</tr>
<tr>
<td>12</td>
<td>Perl-85c-7d</td>
<td>160</td>
<td>7496.618</td>
<td>7379.341</td>
<td>1.56</td>
<td>11</td>
<td>2,4,5</td>
<td>6778.31</td>
</tr>
<tr>
<td>13</td>
<td>Min-27c-5d</td>
<td>2500</td>
<td>3062.017</td>
<td>3062.017</td>
<td>0.00</td>
<td>4</td>
<td>2,3</td>
<td>105.34</td>
</tr>
<tr>
<td>14</td>
<td>Min-134c-8d</td>
<td>850</td>
<td>6287.440</td>
<td>-</td>
<td>-</td>
<td>13</td>
<td>2,3,4,5</td>
<td>TILIM</td>
</tr>
<tr>
<td>15</td>
<td>Daskin-88c-8d</td>
<td>9000000</td>
<td>431.080</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>5,7</td>
<td>TILIM</td>
</tr>
<tr>
<td>16</td>
<td>Daskin-150c-10d</td>
<td>8000000</td>
<td>50234.162</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>1,2,9</td>
<td>TILIM</td>
</tr>
<tr>
<td>17</td>
<td>Or-117c-14d</td>
<td>150</td>
<td>13193.491</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>1,2,5</td>
<td>TILIM</td>
</tr>
<tr>
<td>18</td>
<td>Srivastava-8c-2d</td>
<td>450</td>
<td>472.665</td>
<td>472.665</td>
<td>0.00</td>
<td>2</td>
<td>1,2</td>
<td>2.59</td>
</tr>
</tbody>
</table>

Average Gap 1.07
Median 0.59
For the Perl’s 3 benchmark problems (Perl-12c-2d, Perl-55c-15d and Perl-85c-7d), we compare our results with the results of Perl [64], Hansen et al. [37] and Wu et al. [76]. All the 3 methods are heuristic methods as discussed in the literature. The results are summarized in Table 4-3. It is observed that the solution found for the Perl-12c-2d is the same as the heuristic solutions and the solutions found for Perl-55c-15d and Perl-85c-7d is lower than all the previous results.

In Table 4-4, we compare the results reported in Barreto et al. [12] with our results. In the table provided by Barreto et al. [12], the (IP) column is the best known solution obtained using the clustering heuristic. The lower bound (LB) column was obtained by Barreto [11] with a relaxed 2-index integer linear programming formulation. It was not possible to get LB for the instance Min-134c-8d using the relaxed 2-index integer linear programming formulation. Using the column generation algorithm, a valid LB could not be found in 5 of the instances since the algorithm terminated due to time limit, meaning that the algorithm was continuing to generate promising columns when the time limit has been reached. The optimality is reached in 2 of the 14 instances (Gaskell-29c-5d and Min-27c-5d) by using the clustering heuristic method. In 8 of the 14 instances the proposed integer solutions is lower than the Barreto’s solutions. The solutions obtained in both of the methods are the same in 2 of the 14 instances. In the remaining 4 instances Barreto obtained better results compared to our solutions. In all these 4 instances our algorithm terminated due to time limit. For the instance Christofides-100c-10d, although the proposed algorithm terminated due to time limit, the solution found is better than the Barreto’s solution. Barreto et al. [12] reports that the integrality gap (Gap %) falls between a minimum of 0% and a maximum of 19.01% with

### Table 4-3. Comparison on Perl’s benchmark problems

<table>
<thead>
<tr>
<th></th>
<th>Perl-12c-2d</th>
<th></th>
<th></th>
<th>Perl-55c-15d</th>
<th></th>
<th></th>
<th>Perl-85c-7d</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Depots</td>
<td>Vno</td>
<td>Costs</td>
<td>Depots</td>
<td>Vno</td>
<td>Costs</td>
<td>Depots</td>
<td>Vno</td>
</tr>
<tr>
<td>Perl [64]</td>
<td>1</td>
<td>2</td>
<td>355.58</td>
<td>(2,10,12)</td>
<td>10</td>
<td>5795.62</td>
<td>(2,4,5)</td>
<td>11</td>
</tr>
<tr>
<td>Hansen et al. [37]</td>
<td>(1)</td>
<td>2</td>
<td>355.58</td>
<td>(2,7,12,13)</td>
<td>10</td>
<td>5617.67</td>
<td>(2,4,6)</td>
<td>11</td>
</tr>
<tr>
<td>Wu et al. [76]</td>
<td>(1)</td>
<td>2</td>
<td>355.58</td>
<td>(5,10,12)</td>
<td>10</td>
<td>5532.28</td>
<td>(2,4,6)</td>
<td>12</td>
</tr>
<tr>
<td>Proposed Algorithm</td>
<td>(1)</td>
<td>2</td>
<td>355.58</td>
<td>(2,8,12)</td>
<td>10</td>
<td>5465.51</td>
<td>(2,4,6)</td>
<td>11</td>
</tr>
</tbody>
</table>
an average of 4.81% and a median of 3.13% (In this calculation the instances of Perl are also added). For the same 9 instances solved by both of the algorithms in Table 4-4, the average and the median of gap of Barreto et al. [12] is 3.90% and 2.24% respectively. For the proposed algorithm, the integrality gap (Gap %) falls between a minimum of 0% and a maximum of 4.31% with an average of 1.24% and a median of 0.59%.

Table 4-4. Comparison on benchmark problems that are compiled by Barreto [11]

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>Vcap</th>
<th>IP</th>
<th>LB</th>
<th>Gap (%)</th>
<th>IP</th>
<th>LB</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaskell-21c-5d</td>
<td>6000</td>
<td>435.9</td>
<td>424.9</td>
<td>2.59</td>
<td>424.9</td>
<td>424.9</td>
<td>0.00</td>
</tr>
<tr>
<td>Gaskell-22c-5d</td>
<td>4500</td>
<td>591.5</td>
<td>585.1</td>
<td>1.09</td>
<td>585.1</td>
<td>585.1</td>
<td>0.00</td>
</tr>
<tr>
<td>Gaskell-29c-5d</td>
<td>4500</td>
<td>512.1</td>
<td>512.1</td>
<td>0.00</td>
<td>512.1</td>
<td>505.0</td>
<td>1.39</td>
</tr>
<tr>
<td>Gaskell-32c-5d</td>
<td>8000</td>
<td>571.7</td>
<td>556.5</td>
<td>2.73</td>
<td>568.8</td>
<td>544.3</td>
<td>4.31</td>
</tr>
<tr>
<td>Gaskell-32c-5d</td>
<td>11000</td>
<td>511.4</td>
<td>504.3</td>
<td>1.41</td>
<td>504.3</td>
<td>502.4</td>
<td>0.38</td>
</tr>
<tr>
<td>Gaskell-36c-5d</td>
<td>250</td>
<td>470.7</td>
<td>460.4</td>
<td>2.24</td>
<td>460.4</td>
<td>456.3</td>
<td>0.89</td>
</tr>
<tr>
<td>Christofides-50c-5d</td>
<td>160</td>
<td>582.7</td>
<td>549.4</td>
<td>6.06</td>
<td>565.6</td>
<td>562.3</td>
<td>0.59</td>
</tr>
<tr>
<td>Christofides-75c-10d</td>
<td>140</td>
<td>886.3</td>
<td>744.7</td>
<td>19.01</td>
<td>867.3</td>
<td>835.9</td>
<td>3.63</td>
</tr>
<tr>
<td>Christofides-100c-10d</td>
<td>200</td>
<td>889.4</td>
<td>788.6</td>
<td>12.78</td>
<td>849.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Min-27c-5d</td>
<td>2500</td>
<td>3062</td>
<td>3062</td>
<td>0.00</td>
<td>3062.0</td>
<td>3062.0</td>
<td>0.00</td>
</tr>
<tr>
<td>Min-134c-8d</td>
<td>850</td>
<td>6238</td>
<td>-</td>
<td>-</td>
<td>6287.4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Daskin-88c-8d</td>
<td>9000000</td>
<td>384.9</td>
<td>356.4</td>
<td>8.00</td>
<td>431.1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Daskin-150c-10d</td>
<td>8000000</td>
<td>46642.7</td>
<td>43938.6</td>
<td>6.15</td>
<td>50234.2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Or-117c-14d</td>
<td>150</td>
<td>12474.2</td>
<td>12048.4</td>
<td>3.53</td>
<td>13193.5</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Average Gap** 3.90  **Average Gap** 1.24
**Median** 2.24  **Median** 0.59

### 4.6 Summary and Conclusions

For the Location-Routing Problem, we first introduced a formal MILP model where we have multiple capacitated vehicles and facilities. Then we applied a column generation algorithm tailored to the solution of this kind of LRP. This research is the first study that proposes a column generation algorithm for the solution of the general LRP that has capacitated multiple facilities and vehicles. The LRP has been decomposed into two subproblems. The master problem is a set partitioning problem with side constraints and the subproblem is an elementary shortest path problem with resource constraints. Column generation technique has been applied to solve the relaxed master problem (RMP) and a new dynamic programming algorithm has been designed in order
to generate the columns in the subproblems. The gaps obtained are tightened since we solve an elementary shortest path problem hence eliminate all the k-cycles. It is observed that linear master problem is solved easily. Although the running times for the master problem increase as the number of columns increase, the total running times spent in solving the relaxed master problems is always negligible compared to solution times required for the subproblems. Hence, solving the subproblems efficiently is very crucial for a good column generation algorithm. The column generation algorithm has been successful in solving instances up to 10 candidate facility locations and 100 customers. For the larger instances the number of labels created at each solution of the subproblem has been very large and the running times to find a set of good routes increased. Therefore, the algorithm ran till the time limit for 5 of the 18 instances.

Further work for the early elimination of infeasible or unnecessary labels is required to solve larger instances efficiently. For instance, adding a more sophisticated dominance criterion should improve the efficiency of the dynamic programming procedure hence affect the overall performance of the column generation algorithm. Future work should also focus on adding cutting planes to the master problem to allow larger instances to be solved.

During the execution of the column generation algorithm we focused to update bounds frequently and keep the gaps between the upper and lower bounds in a manageable level. It is confirmed in the literature that it is very crucial to have small gaps to solve the integer programs efficiently and computational experiments revealed that the strategy applied in the algorithm has resulted into small gaps. In our computational experiments, the integrality gap (Gap %) falls between a minimum of 0% and a maximum of 4.31% with an average of 1.18% and a median of 0.89%.

Benchmark problems are very important to check the solution quality and efficiency. Barreto’s effort to set a widely accepted LRP benchmark problem sets is worthy. Benchmarks are crucial to make comparisons with the peer papers and it lets the
researchers to test their solution quality and make inferences on the efficiency of their solution method. The number of papers that make comparisons with each other will probably increase for the general capacitated LRP as the benchmark problems for this kind of LRP are accepted and studied more in the near future.

As a future work, adding time window to the LRP problem will make it more close to the real life situations considering that some customers often impose service deadlines and desirable service time restrictions. The time window extension will be especially important with the advent of Just-In-Time (JIT) logistics principle. Min et al. [57] reports that only five LRP articles (less than 16%) account for hard time windows and none considers soft time windows. If a LRP with time windows is solved, the incompatible pair and path cuts that is introduced by Bard et al. [9] for the vehicle routing problem with time window can be incorporated to the master problem of the column generation algorithm.
CHAPTER 5
SOLVING THE CURFEW PLANNING PROBLEM

5.1 Introduction

Curfew planning is an allocation of resources for the annual maintenance of the railway network of railroads that regulates the number of disruptions while honoring several requirements. A region is said to be under curfew if it is under active maintenance resulting in a blockage or train schedule disruption. Curfew planning is important because of the steep growth that railway transportation has undergone over the last decade due to economic expansion and resulting road congestion and pollution problems [32]. In 2001, the freight traffic in ton-miles by rail was 47% in comparison to trucks’ 33% and ships’ 20% [73]. Rail transport also diverts some freight traffic from the roads, partially alleviating congestion and wear and tear on highways [55]. Its growth has guided the infrastructure development and generated the need for strategic and optimized maintenance operations that honor safety standards [19]. The Curfew Planning Problem (CPP) is one of such operations encountered by railroads when maintaining their railway tracks. It consists of optimally assigning crews to tracks so that all maintenance tasks are completed with a minimum disruption of the regular train schedules.

There are two versions of this problem: continuous and intermittent. In the continuous CPP, once the crews start working on tracks, the tracks become unavailable for operations until the job is finished. In the intermittent version of the problem, crews stop and resume the operations as per the train schedules. The continuous version of the problem is harder since an entire planning period needs to be divided into continuous time-blocks. It can easily be proved NP-Complete by showing resource allocation problems as its special case. The intermittent curfew planning problem is relatively simple as the restriction of finding continuous time-blocks is relaxed. In this
paper we study the continuous curfew planning problem; however, the same concept can be applied to modify the model used for capturing the intermittent version.

The continuous curfew planning problem considers the optimization of the existing infrastructure combined with ordinary and renewal maintenance techniques. Grimes and Barkan [36] explain the differences in these two techniques while suggesting the cost-effectiveness of renewal maintenance. In Australian freight operations, the maintenance costs comprise between 25% and 35% of total train operating costs (1998). A study on the subway of Hong Kong suggests the criticality of the problem by presenting the real-time data of daily ridership and time slots available for maintenance [18]. The U.S railroads invest around $15 billion for annual maintenance [2] and even marginal improvement will save millions every year. All of these studies show the global and economic importance of the problem. Additionally, there is a broad scope of improvement due to more developed infrastructure, better planning, and better strategic decisions [28].

The railway network of railroads usually spans an entire country, and the tracks are divided into several regions and terminals. At the beginning of the year, the project manager evaluates the network and prepares the list of tracks to be repaired or replaced. These tracks are referred to as jobs, and they fall into seven categories: (a) Capacity Jobs, (b) Curve Patching, (c) Concrete Repair, (d) Gauging, (e) Out-of-Face-New, (f) Out-of-Face-Repair, and (g) Tie and Surfacing. These jobs are grouped into two main categories: Rail jobs (a to f), and Tie jobs (g). The jobs in a particular region are quite close to each other and collectively are called a project. All rail jobs of a region are grouped in a rail project and tie jobs are grouped in a tie project. Sometimes regions also are referred to as subdivisions. Inside a subdivision there is usually (but not necessarily) a rail project and a tie project. Some projects can be completed in the railroad yards and don't disrupt the regular train schedule, while some have to be performed on the mainline. The mainline can be single-tracked or double-tracked. The
region with projects on a single-tracked mainline are said to be *under absolute curfew*, while the rest of them are *under normal curfew*. It is imperative that the absolute curfew is far more restrictive than the normal curfew. In some cases, absolute curfew is avoided by changing the infrastructure (sidings) on single-tracked segments.

The duration of a rail project is calculated by summing up the track distance being repaired or replaced, while that of a tie project is found by adding the number of ties being processed. It also depends on the type of crews working on the project. Crews are classified in four categories: (i) Large-rail crews (preferred for job types *e* and *f*), (ii) Small-rail crews (preferred for job types *b*, *c*, and *d*), (iii) Large-tie crews, and (iv) Small-tie crews. Durations also depend on the configuration (categories) of jobs which constitute the project. For example, a small-rail crew is more specialized in curve-patching jobs, while a large-rail crew concentrates on out-of-face jobs. A large crew typically consists of double the number of members as in a small crew. Hence, the ratio of the time taken by small crews to that of larger crews is approximately (but not necessarily) 2:1. A project also can be crashed to complete it faster. *Assigning multiple crews on a project to decrease its active period is called crashing*. To keep the problem size mathematically tractable, we assume the week as the time unit.

After completing the projects in it, the crews travel from one region to the next within a maximum distance limit, which usually is 400 miles. When handling special projects, *jamboree projects*, we relax the maximum distance restriction. These projects have a fixed earliest starting time and a latest finishing time called *jamboree weeks* and are forced by special situations such as a vacation in coal mines, a football season, etc. These projects must be completed within the jamboree weeks as the cost of violations is very high. The restriction of continuity also can be relaxed but with a very high penalty. In our proposed models, we cover all of these issues along with the other restrictions discussed in the Section 5.2.
Although the basic problem has long existed, the amount of literature devoted to it is very scarce. The studies performed so far help us to understand its criticality and complexity. A brief literature survey is presented in Section 5.3. In this paper we consider the global problem, including all real-life attributes, and discuss some holistic approaches. These approaches are the first attempts at capturing and solving all issues in one model, so that they can be implemented in real-life scenarios. We propose three exact algorithms which vary in their capabilities and effectiveness for handling different aspects of the problem. We also suggest a decomposition-based heuristic which generates practical results within a reasonable time-frame. Our models consider all major factors affecting the efficiency of the curfew plan and are flexible enough to be extended for capturing new requirements. In Section 5.4, we formulate the CPP as a Mixed Integer Programming (MIP) Problem on an appropriately defined time-space network. This model is useful for understanding the intricacies and complexities of the problem. The contributions of this paper include:

1. We present a unique, duty-generation model for the CPP. Its framework captures all issues discussed earlier, and it is very flexible and compact (Section 5.5).

2. We propose a column-generation approach to improve the solutions obtained from the duty-generation model. Although it is partially successful in the current scope of the problem, it generates good solutions in the scenario where project crashing is not allowed (Section 5.6).

3. We suggest a decomposition-based heuristic that generates very practical and excellent quality solutions which can be implemented directly into real-life applications. The computational results on real-world instances taken from a large railroad show a significant improvement in the number of disruptions or violations (Section 5.7).

5.2 Problem Description

The primary goal of the Curfew Planning Problem (CPP) is to solve the decision version of the optimization problem that finds a feasible solution under several restrictions, which are discussed next in this section. Optimizing the feasible solution,
minimizing the inter-project distance travelled by all crews, is a secondary goal. Before introducing the mathematical model, we describe the business constraints in detail:

1. **Crew Continuity Restriction**: The goal is to have all crews working without any break for the entire year, except during the training periods. This restriction can be relaxed for the last few working weeks if there aren’t any unfinished projects.

2. **At-Most Absolute Curfew**: The large amount of freight volume transported by railroads allows them to cut off only a certain number of subdivisions. As stated earlier, a subdivision is said to be cut off if it is under absolute curfew. All projects inside a subdivision are counted as one absolute curfew if they are active in the same week.

3. **Precedence Restriction**: Rail projects must be completed before the tie projects in a segment which has been scheduled for both types of maintenance. This precedence reduces the wear and tear of ties in the construction or maintenance phase.

4. **Service-Corridors**: A service-corridor is the group of railway track segments (subdivisions) that share the majority of the freight transportation. Ideally, only one of the subdivisions in each service corridor should be under absolute curfew.

5. **Mutually exclusivity**: The nature of the railway transportation and the load shared by it simultaneously restricts several groups of regions to be under absolute curfews. These restrictions coupled with service-corridor restrictions significantly reduce the feasibility region.

6. **Time Windows**: Each project is allowed to be active only within a pre-specified time window. A project is said to be active if a crew is working on it. These restrictions arise due to seasonal factors such as extreme cold, storms, and games, etc. They have a mixed effect on the models, as they can be utilized to reduce the problem size but at the cost of significantly reducing the feasible region.

As discussed in the previous section, there are some exceptional (jamboree) projects for which these restrictions can be relaxed with a predefined penalty. If jamboree projects are not completed in their given time window, the financial loss is much higher than the penalty occurred due to the violation of the restrictions listed above. The complete problem in its pure form is very complex, and most of the previous studies have tackled only a part of the problem.
5.3 Literature Review

Our brief survey of literature is devoted to describing various versions of the curfew planning problem (CPP). One of the very first studies focuses on railway maintenance operations and was performed by A. Higgins [39]. It proposes a model aimed at determining the best allocation of maintenance activities and crews for minimizing the disruption in the scheduled trains and reducing the completion time. He considers a subset of the previously described constraints and gives a non-linear formulation of the problem. The non-linearity and large size of the problem force the use of heuristics and hence a tabu-search heuristic has been proposed that generates the neighbors by swapping the ordering of jobs, crews, or both. Higgins et al. [39] use this model for the computational testing on an 89-km. track corridor and improve the manually created schedules by almost 7%. A dynamic schedule generation technique for the rolling horizon is discussed by Bruce et al. [18]. Their study is based on real-time data from the Hong Kong subway railway system. Lake et al. [47] have done studies on Australian freight operations. They say that railways operate under the conflicting objective of minimizing the infrastructure costs while maintaining or improving the service qualities. These conflicts have intensified due to industrial growth, high traffic volume, and other related factors. They develop the model for the short-term maintenance scheduling of track-maintenance activities after the train schedules and maintenance activities have been planned. Lake et al. [47] solve it using a two-phase heuristics technique, with the first stage generating a feasible solution, and the second stage implementing simulated annealing. They apply these algorithms on twenty-track segments and on a seven-day scheduling period.

For occasionally used tracks, e.g. in Australia and some European countries, Budai et al. [19] show that the track-possession is modeled in-between operations. They introduce a slightly different version of the problem where the main purpose is completing the project within the track’s free time. It creates a dynamic schedule for
carrying out preventive maintenance activities. It then proposes three heuristics to tackle the problem under several limitations, and the heuristic called Max-to-Min is found to be the most successful for this problem. The basic concept of this heuristic is to consider projects based on their frequencies of occurrence and find the best starting times such that the number of projects being jointly scheduled is maximized. Grimes and Barkan [36] perform a study to measure the cost-effectiveness of railway infrastructure renewal maintenance. Their results show that if railroads constrain the renewal maintenance to reduce the overall capital expenditures, increasing maintenance expenses will more than offset temporary reductions in capital spending. A detailed survey of maintenance operations is done by Oke [60] which describes the current state of the art in this field.

To summarize, the algorithms developed thus far to solve the CPP only considers a part of the problem but have been very helpful in preparing the foundation of our work. We consider the optimization problem of assigning the crews to projects, such that primarily all restrictions are satisfied and secondarily the distance travelled by crews between projects during the weekends is minimized. In this paper, we solve the problem using a time-space network formulation (TSNF), the duty-generation model (DGM), the column-generation model (CGM), and decomposition-based heuristics. We also suggest several extensions for each model that can further strengthen them in special cases.

The problem of assigning the crews to the projects is itself a challenging problem, because of its combinatorial nature. It easily can be shown to be NP-Complete, as the resource allocation problem is its special case and has been proven to be a hard problem [33]. In this paper, we solve the continuous version of the problem where projects have to be completed once started, excluding the special cases (jamboree projects) discussed earlier. We now propose the time-space network formulation of the problem.
5.4 Time-Space Network Formulation (TSNF)

The main variables in the decision version of the CPP are used to find the starting time of each project, as well as the type and number of crews being assigned to it. These variables easily can be combined to generate the annual schedule (timetable) for the crews. In the basic TSNF, we combine the decision factors and generate a multi-commodity flow network. The type of crews is referred to as a commodity.

The basic network contains a starting node and a finishing node for each project in each week of the working year. Two arcs emanate from each starting node of a project: one indicating the flow of a small crew and the other that of a large crew. It is imperative here that from a rail (tie) project, only the arc corresponding to rail (tie) crew will emanate, because rail (tie) projects only can be done by rail (tie) crews. We call them project arcs. The length of the arc represents the time taken to complete the project. Arcs called connection arcs connect the finishing nodes of each project in each week with the starting nodes of the projects of the same week that fall within the maximum distance limit. We also introduce a source node from which all crews start and a sink node to which they all finish. The source node is connected to the first starting node, and the sink node is connected the last finishing node of each project for each crew-type.

Figure 5-1 shows a simplified network where the tail nodes of the colored arcs are starting nodes and the head nodes are finishing nodes. The blue arc corresponds to the small crews and the red arc to the large crews. Every arc is of unit capacity. There are some extra nodes, called pool nodes and indicated by $A_t$, which have been introduced to decrease the size of the network. We will discuss them later in this section. The time factor of the network is along the y-axis and the space factor is along the x-axis in Figure 5-1.

To reduce the number of connecting arcs, we introduce the concept of the relocation pools. A relocation pool is a place where crews go after finishing a project and from
there get relocated to either the starting node of a project or directly to the sink node. We create several relocation pools in each week based on the geographical structure of the network and the job’s volume. All starting nodes and finishing nodes are connected through arcs, called relocation arcs, to the relocation pools of the same week which are within a predefined factor of maximum distance limit based on previous experiments. Although the relocation pools reduce the exactness of the problem, they make the problem tractable. In our formulation, a crew may relocate from project $p$ to project...
only if there exists a pool connected to both projects. Specifically, if project \( p \) is connected to pool \( A \), then each finishing node \( p_j \) will be connected to the relocation pool node \( A_j \) by a relocation arc in week \( j \), excluding the last week. Similarly, the starting node of every project will be connected to all pool nodes of the same week, which satisfies the connectivity conditions excluding the first week. We exclude the first and the last week’s arcs as they are redundant.

If the restrictions mentioned in Section 5.2 can be relaxed, the problem can be expressed as a linear program. The solution of this linear formulation would always result in an integer optimal solution due to the total unimodularity of the constraint matrix. Unfortunately, the side constraints make this problem very difficult. However, we can reduce the number of integer variables by defining everything linear except for the variables corresponding to the flow on the project arcs. The flow conservation constraints mixed with the objective function force all other variables to be integers.

To increase the feasibility of the problem and find the initial solution easily, we relax the continuity restrictions and introduce sojourn arcs between contiguous relocation-pool nodes (see Figure 5-1). This allows the crews to remain in the pool and be assigned later in the project span. But we discourage these flows by assigning very high costs on these arcs. To further improve the feasibility, we can allow crews to travel from a relocation pool to another at very high costs, since it will introduce many infeasible travels during weekends. Please note that relocation-pool nodes for each project in each week appear twice in the network (Figure 5-1) to make the diagram cleaner, though they are the same in the original network. Next we show the mathematical model for the time-space network.

5.4.1 Mathematical Model

Index:

\( w \): Week.
\( k \): Crew-type.
$p$: Project.

$u$: Service corridor.

$s$: Subdivision.

$n$: Node of TSNF.

$e$: Arc of TSNF.

**Parameters:**

$P$: Set of all projects.

$PR$: Set of all rail projects.

$PT$: Set of all tie-surfacing projects.

$\lambda^s_u$: 1 if subdivision $s$ is in service corridor $u$, otherwise 0.

$\gamma^p$: 1 if project $p$ requires an absolute curfew, otherwise 0.

$K$: Total types of crews.

$\alpha^k$: Number of available crews of type $k$.

$W$: Total number of weeks.

$P^{\text{prec}}$: Set of projects under precedence constraints.

$S^{\text{adj}}$: Set of mutually exclusive (adjacent) subdivisions.

$U$: Set of service corridors.

$R$: The set of relocation pools.

$A$: Set of arcs in the network.

$A(w)$: Set of all arcs which cross the week $w$.

$N$: Set of nodes in the network.

$c^k_e$: Cost incurred if crew-type $k$ is assigned on arc $e \in A$.

$\mu$: Number of absolute curfews allowed per week.

$I_{ps}$: 1 if project $p$ is in subdivision $s$.

$S$: Set of subdivisions.

$M_s$: Number of projects that require absolute curfew in subdivision $s$. 
\( \beta_s \): 1 if subdivision \( s \) is on a single track.

\( c_{w^{\text{curf}}} \): Cost of violation in at-most-curfew constraints in week \( w \).

\( c_{w^{\text{subm}}} \): Cost of violation in mutually exclusive constraints in week \( w \).

\( c_{w^{\text{serv}}} \): Cost of violation in service corridor constraints in week \( w \) for corridor \( u \).

\( c_{pq}^{\text{prec}} \): Cost of violation in precedence constraints for pair \( pq \).

**Decision Variables:**

\( x_{pw}^{k} \): 1 if project \( p \) starts in week \( w \) by a crew of type \( k \); 0 otherwise \( \forall p \in P, k \in K, w \in W \). We also use the symbol \( x_e^k \) for better understanding, where each arc number is a combination of \( p \) and \( w \). \( x_{pw}^k \) is used only for project arcs while \( x_e^k \) is used for all arcs.

\( y_{sw} \): 1 if subdivision \( s \) is under absolute curfew in week \( w \); otherwise 0.

\( v_{w}^{\text{curf}} \): Violation in at-most-curfew constraints in week \( w \).

\( v_{s1s2w}^{\text{subm}} \): Violation in mutually exclusive constraints in week \( w \).

\( v_{uw}^{\text{serv}} \): Violation in service corridor constraints in week \( w \) for corridor \( u \).

\( v_{pq}^{\text{prec}} \): Violation in precedence constraints for pair \( pq \).

**Objective:**

Minimize

\[
\sum_{e \in A} \sum_{k \in K} c_e^k x_e^k + \sum_{w \in W} c_{w}^{\text{curf}} v_{w}^{\text{curf}} + \sum_{s1,s2 \in S^{\text{ME}}} \sum_{w \in W} c_{w}^{\text{subm}} v_{s1,s2,w}^{\text{subm}} + \sum_{u \in U} \sum_{w \in W} c_{uw}^{\text{serv}} v_{uw}^{\text{serv}} + \sum_{pq \in P^{\text{prec}}} c_{pq}^{\text{prec}} v_{pq}^{\text{prec}}
\]

**Constraints:**

\[
\sum_{e \in I(n)} x_e^k - \sum_{e \in O(n)} x_e^k = 0 \quad \forall n \in N/\{s, t\}, k \in K \tag{5-1}
\]

\[
\sum_{e \in A(w)} x_e^k \leq \alpha_k \quad \forall w \in W, k \in K \tag{5-2}
\]

\[
\sum_{w \in W} \sum_{\nu \in K} x_{pw}^k = 1 \quad \forall \nu \in P \tag{5-3}
\]

\[
\sum_{p \in P} \sum_{k \in K} \gamma_p^k x_{pw}^k \leq M s y_{sw} \quad \forall w \in W, s \in S \tag{5-4}
\]

\[
\sum_{s \in S} y_{sw} \leq \mu + v_{w}^{\text{curf}} \quad \forall w \in W \tag{5-5}
\]
\[
\sum_{k \in K} \sum_{w \in W} (w + 1)x_{pw}^k - \sum_{k \in K} \sum_{w \in W} (w)x_{qw}^k \leq v_{pq}^{\text{prec}} \quad \forall (p, q) \in P^{\text{prec}} \tag{5-6}
\]

\[
\sum_{s \in S} \beta_{su}^s \lambda_u^s y_{sw} \leq 1 + v_{uw}^{\text{serv}} \quad \forall w \in W, u \in U \tag{5-7}
\]

\[
y_{s1w} + y_{s2w} \leq 1 + v_{s1,s2,w}^{\text{subm}} \quad \forall w \in W, m \in S^{\text{ME}} \tag{5-8}
\]

\[
x_{p,w}^k \in \{0, 1\} \quad \forall k \in K, p \in P, w \in W \tag{5-9}
\]

The objective function simply minimizes the cost of arcs explained earlier, as the main focus is on finding a feasible solution. Constraints 5–1 maintain the balanced flow of each crew-type throughout the network. Constraints 5–2 keep a check on the number of crews employed while constraints 5–3 state that each project must be completed by exactly one crew. Constraints 5–4 control the value of subdivision variables. Constraints 5–5 enforce the maximum absolute curfews allowed per week, and Constraints 5–6 track the precedence relations. Constraints 5–7 maintain the service-corridor absolute curfew restrictions. Constraints 5–8 do not allow two mutually exclusive subdivisions to become active simultaneously. Constraints 5–9 are the integrality constraints, although many of these can be relaxed to linear variables as discussed earlier in the section.

### 5.4.2 Computational Analysis

The time-space network formulation (TSNF) has some implicit advantages, specifically: (i) All variables are implicitly confined to the unit hypercube, and (ii) Time-window constraints are implicitly enforced as we don’t draw the corresponding arcs. The main limitation of this approach is that it does not allow project crashing. We consider two real-life problems of a major railroad with 19 crews and 300 projects that have rail and tie projects almost equally divided. We took 50 weeks as the available number of weeks in a year, as two weeks are generally the training/off weeks. There are usually ten service-corridors, each containing a group of subdivisions. The project completion times, time windows, precedence relations, and other data were collected directly from railroads. The details of instances are given in the computational sections.
We use the same datasets for testing each formulation. The order of number of variables are \(O(|P|^*W^*|k|/2^*r)\), where \(r\) is average number of projects which are within the maximum distance limit of each other. The order of number of constraints are \(O(W^*(1+|P|+k+2|S|+|U|+|P|))\). For the real-life instances, the average number of variables was around 600,000, and the average number of constraints was around 65,000.

We solve this formulation using CPLEX 11.2. The basic model without using the relocation pool didn’t give any solution within 24 hours. We didn’t allow the process to run for more than 24 hours, because it loses the practicality of the approach. A time limit of eight hours is good from the practical point of view. The introduction of relocation pools significantly improves the solution time to two hours, but the number of travels violating the maximum distance limit increased to an unacceptable limit.

The rail projects and tie projects can be divided and solved independently and finally combined to create a global solution. The process of combining needs some re-optimization, as some constraints get violated when the independent solutions are merged. We solve these decomposed problems and are able to find the feasible solutions for the same problem instance within five minutes for each part. The process of combining takes another 15 minutes, but it results in several other violations with penalty. Though this model is not very successful, it is good for better understanding the problem and identifying the critical issues. The main drawback of the model is its incapability in handling project crashing without which both real-life instances cause large numbers of violations. Project crashing helps reduce the number of absolute curfews in a week, and thus improves the feasibility. To include it, we would have to introduce another set of arcs for each project, and that would make the model even more intractable. These results guide us toward developing a more compact formulation that can handle the project crashing and other constraints more efficiently. We discuss this model, the duty-generation model, in the following section.
5.5 Duty-Generation Model (DGM)

The duty-generation model (DGM) is an MIP approach with the primary focus on finding the feasible solution that couldn’t be obtained efficiently by the first model. The incapability of project crashing, large number of variables and constraints were the main drawbacks of the time-space network, because they increased the running time and reduced the practicality as well as quality of the solution. All of these guided us toward a stronger and more compact formulation. In this model, we generate several duties for each project. A duty indicates the starting and finishing week of the project and the type of crew by which it is being processed. It is a column of zeros and ones, where the first occurrence of one gives the starting week and where the last occurrence gives the finishing week of the project. Each duty can be seen as the potential schedule of the corresponding project. To maintain consistency, we keep the parameters’ definitions intact as given in the first model and introduce new symbols for additional parameters.

We create a matrix \( D[p] \) for each project \( p \) which is a collection of all candidate duties of the project \( p \). The duties are identified by their unique id. In this formulation, we have the flexibility of crashing the projects by combining two small crews or two large crews. This feature cannot be implemented in the first model, as each new crew-set doubles the size of the problem. A crew-set is the combination of crews employed to complete a project. We create four crew-sets for each crew-type (rail and tie): (i) 1-small crew, (ii) 1-large crew, (iii) 2-small crew, and (iv) 2-large crew; where the last two are

<table>
<thead>
<tr>
<th>Crew Set 1</th>
<th>Crew Set 2</th>
<th>Crew Set 3</th>
<th>Crew Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 . . 0 1 0 . . 0 1 0 . . 0 1 0 . . 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 . . 0 1 1 . . 0 1 1 . . 0 0 1 . . 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 . . 0 1 1 . . 0 0 1 . . 0 0 0 . . 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 . . 0 0 1 . . 0 0 0 . . 0 0 0 . . 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>. . . . . . . . . . . . . . . . . . . .</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 . . 1 0 0 . . 1 0 0 . . 1 0 0 . . 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 . . 1 0 0 . . 1 0 0 . . 1 0 0 . . 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-2. Variable matrix \( D[p] \) for project \( p \)
employed for project crashing. An example of a duty matrix is given in Figure 5-2, where the project takes four, three, two, or one week to complete if done by the four crew-sets discussed above, respectively. Next we present the mathematical formulation:

**Index:**

d: Column of matrix D.

\(a_{wd}\): Value at \(w\)th row and \(d\)th column of matrix D.

c: Element of a crew-set CS.

**Parameters:**

D: Set of duties for all projects.

\(D[p]\): Set of duties for project \(p\).

\(D[p][c]\): Set of duties for the project \(p\) done by crew-set \(c\).

\(\sigma_d^k\): Number of crew-type \(k\) in duty \(d\).

\(\gamma_d\): 1 if duty \(d\) requires an absolute curfew, otherwise 0.

\(l_{ds}\): 1 if duty \(d\) is in subdivision \(s\), otherwise 0.

\(\lambda_d^u\): 1 if duty \(d\) is in service corridor \(u\), otherwise 0.

\(f_d\): Starting week of duty \(d\), i.e. the week in which first 1 comes in column \(d\).

\(l_d\): Finishing week of duty \(d\), i.e. the week in which last 1 comes in column \(d\).

\(ct[d]\): crew type involved in duty \(d\).

**Variables:**

\(x_d = 1\) if \(d\)th column of matrix D is the duty-schedule for its project; 0 otherwise.

\(y_{sw}\): 1 if subdivision \(s\) is under absolute curfew in week \(w\); otherwise 0.

\(\nu_{w}^{curf}\): Violation in at-most-curfew constraints in week \(w\).

\(\nu_{s1s2w}^{subm}\): Violation in mutually exclusive constraints in week \(w\).

\(\nu_{uw}^{serv}\): Violation in service corridor constraints in week \(w\) for corridor \(u\).

\(\nu_{pq}^{prec}\): Violation in precedence constraints for pair \(pq\).
\( \nu_{d}^{dist} \): Violation in partial distance constraints.

The duty-generation model (DGM) is solved in three phases. Each phase takes the previous output as the input and then optimizes the local objectives contributing toward the global ones.

5.5.1 Duty-Generation Phase 1: Variable Reduction

This phase is short but very important from the implementation point of view, because it reduces the problem size by removing all duties which either violate time windows or are too costly. The cost of a duty is governed by its starting time and finishing time, along with the crews with which it is associated. A duty is costly if it violates many constraints. We handle the time-window constraints in this phase by permitting only those duties which are within the acceptable (with penalty) limit. In our problem instances, we allow two weeks violations at the most. We also may change the cost of duties in this phase as required in the specific cases, such as fixing a specific project to be completed preferably by a small team. We do not create duties, which indicate project crashing by small (or large) crews, for the projects whose duration is one week. In the next phase, we formulate the problem as an MIP.

5.5.2 Duty-Generation Phase 2: Project Scheduling

This phase is the most crucial for the DGM, as all of the critical decision variables are defined here. The modified matrix generated by Phase 1 is used to build all constraints.

**Objective:**

Minimize

\[
\sum_{d \in D} c_d x_d + \sum_{w \in W} c_w^{curf} \nu_w^{curf} + \sum_{s_1, s_2 \in S^{ME}} \sum_{w \in W} c_w^{subm} \nu_{s_1, s_2, w}^{subm} + \sum_{u \in U} \sum_{w \in W} c_{uw}^{serv} \nu_{uw}^{serv} + \\
\sum_{pq \in P^{prec}} c_{pq}^{prec} \nu_{pq}^{prec} + \sum_{dd \in D} \nu_{d}^{dist}
\]
Constraints:

\[
\sum_{c \in CS} \sum_{d \in D[p,c]} x_d = 1 \quad \forall p \in P \tag{5-10}
\]

\[
\sum_{d \in D} \sigma^k_d a_{wd} x_d = \alpha^k \quad \forall w \in W/\{50\}, \ k \in K \tag{5-11}
\]

\[
\sum_{d \in D} \sigma^k_d a_{wd} x_d \leq \alpha^k \quad \forall w = \{50\}, \ k \in K \tag{5-12}
\]

\[
\sum_{d \in D} \sum_{d : \{f_d \leq w \leq l_d\}} I_{ds} \gamma_d a_{wd} x_d \leq M_s y_{sw} \quad \forall s \in S, w \in W \tag{5-13}
\]

\[
\sum_{s \in S} y_{sw} \leq \mu + \nu_w^{curf} \quad \forall w \in W \tag{5-14}
\]

\[
\sum_{c \in CS} \sum_{d \in D[p,s]} a_{wd} x_d + \sum_{c \in CS} \sum_{d \in D[q,s]} a_{wd} x_d \leq 1 + \nu^{prom}_{p1,p2,w} \quad \forall w \in W, (p, q) \in P^{ME} \tag{5-15}
\]

\[
y_{s1w} + y_{s2w} \leq 1 + \nu^{subm}_{s1s2w} \quad \forall w \in W, m \in S^{adj} \tag{5-16}
\]

\[
\sum_{c \in CS} \sum_{d \in D[p,s]} f_d x_d - \sum_{c \in CS} \sum_{d \in D[q,s]} f_d x_d \leq 1 + \nu^{prec}_{pq} \quad \forall (p, q) \in P^{prec} \tag{5-17}
\]

\[
x_d \sum_{s \in S} \beta_s \lambda^s_{sw} y_{sw} \leq 1 + \nu^{serv}_{aw} \quad \forall w \in W, u \in U \tag{5-18}
\]

\[
x_d - \sum_{d' : \{f_{d'} = l_d + 1, dist[d',d] < \text{MAXDIST}, ct[d] = ct[d']\}} x_{d'} \leq 0 + \nu_{d}^{dist} \quad \forall d \in D, l_d \leq 45 \tag{5-19}
\]

\[
x_d - \sum_{d' : \{f_{d'} = f_d - 1, dist[d',d] < \text{MAXDIST}, ct[d] = ct[d']\}} x_{d'} \leq 0 + \nu_{d}^{dist} \quad \forall d \in D, f_d \leq 45 \tag{5-20}
\]

\[
x_d \in \{0, 1\} \quad \forall d \in D \tag{5-21}
\]

The objective function minimizes the costs, which include the duties containing undesirable characteristics, and the penalty for the constraint violations. Constraints 5–10 ensure the completion of each project by exactly one crew set. Constraints 5–11 and 5–12 maintain the continuity restriction in all weeks, except for the last one as we relax the constraint in the last week. As we discussed in time-space network model,
Constraints 5–13 control the subdivision variables assignment. Constraints 5–14 keep check on the number of the absolute curfews in each week. Constraints 5–15 and 5–16, respectively, do not allow mutually exclusive projects and subdivisions to be active simultaneously. Constraints 5–17 are for precedence relations among the projects, and Constraints 5–18 maintain the service corridor restrictions. We try to partially enforce the distance constraints through set 5–19 and 5–20. The DGM has been developed at the crew-type level, so there is no direct way to force the maximum distance limit. The basic concept is that for each duty ending in week \( w \), there should be at least one duty containing at least one of the same crew-types within the allowed distance limit starting in week \( w + 1 \); and for each pair of duties ending in week \( w \), there should be at least two duties starting in week \( w + 1 \), and so on. Setting all of these as constraints would have made the problem intractable. So we try a different method to partially force these constraints. In Constraints 5–19, we state that for each duty ending in week \( w \), there should be at least one duty containing at least one of the same crew-types within the distance limit starting in week \( w + 1 \). In Constraints 5–20, we say that for each duty ending in week \( w \), there should be at least one duty containing at least one of the same crew-types starting in week \( w − 1 \). These sets of constraints together handle the distance constraints very effectively, as shown in the computational section (Section 5.8). These two sets of constraints are referred to as partial distance constraints throughout the paper. We relax these restrictions for last five weeks for two reasons: (i) to improve the feasibility, and (ii) because it might be the last project done by the crew. Set 5–21 are the integrality constraints. The output of this phase is the starting week and crew-type employed for each project, and they are used in the next phase to generate the crew schedules.

5.5.3 Duty-Generation Phase 3: Crew-Scheduling

The project-scheduling phase is based on the crew-type and gives the schedule of projects. We need to find the working plan of each crew, which is done in the
crew-scheduling phase. Using the output of Phase 2, it constructs four minimum-cost-flow-problems (MCFP) and determines the sequence of projects for each crew such that the total distance travelled is minimized. Each MCFP belongs to one of the four crew-types (small-rail, large-rail, small-tie, and large-tie) and finds the flow of each crew through projects. The underlying network for the crew-type $k$ is created as follows:

1. **Nodes creation:** Create a node for each project, which is done by crew-type $k$.
2. **Arcs creation:** An arc between node $p$ and $q$ exists if (i) the finishing week of $p$ is one week before the starting week of $q$, and (ii) inter-project distance of $p$ and $q$ is less than the maximum-distance-limit (MAXDIST). In our model, we introduce the arcs only if the first condition is satisfied, as all good feasible solutions have some distance violations.
3. **Arc costs:** The arc cost is a function of the inter-project distance, except for the projects done in jamboree and year-end weeks.

Each independent problem can be solved using well-studied minimum-cost-flow algorithms [5]. As all side constraints are satisfied in the project scheduling phase, optimizing the flow over a network will generate the best crew-schedules.

**5.5.4 Computational Analysis**

Phase 2 of the DGM is the main optimization phase where most of the time was spent, since Phase 1 and Phase 3 have polynomial complexities. In the project scheduling phase (Phase 2), the order of the number of integer variables is $O(|P|^*W^{*}k/2)$ and that of the constraints is $O(|P|^*W(1 + k + 2|S| + |U| + |P|^*k/2))$. For the instances being considered, these values are around 50,000 and 80,000 – and there are 80,000 other auxiliary penalty variables, one for each performance constraint. We tested the DGM on the same problem instances and found that even the partial distance constraints are bottlenecks. So, we relaxed these constraints and solved the model using CPLEX 11.2. The relaxed model was very effective and time efficient. It generated good-quality solutions within an hour, but the number of distance violations was very high. We tried to capture the distance constraints in Phase 3, where the arcs connecting the projects out of the distance limit are heavily penalized. It solved each MCFP within
seconds, but ignoring the distance constraints in second phase proved costly to the solutions’ quality. Phase 2 generated solutions that restricted the feasibility region of Phase 3 significantly. We discuss these details in the Section 5.8.

The main advantage of the DGM is its flexibility in easily including new business requirements. The solutions generated are very good, except for the distance violations. This behavior encouraged us to develop the column-generation model, which can improve these solutions.

5.6 Column-Generation Model (CGM)

The column-generation approach is widely used for problems with a large number of variables whose linear relaxation generates the fractional value for most of the variables [10]. The details of this approach can be studied in the paper by Lübbecke and Desrosiers [53]. It starts with an initial feasible solution obtained by the DGM or local heuristics and tries to improve it. The solutions obtained by the DGM have several distance constraints violations, while the initial solution for the CGM must be feasible. Since all constraints are soft and we aim to minimize the penalty, the solutions of the DGM can be used as the starting solution. So we relax these constraints as we did earlier by using penalty structures.

The column-generation approach has three building blocks: (i) the generation of initial schedules, (ii) the formulation of the master problem, (iii) and the development of sub-problem structures. We can find the initial schedules using several simple local heuristics. Single project schedules are the simplest to start with, but these end up having a very high cost. We generate routes by assigning an unscheduled project to the next available crew. Some potential routes also are taken from the output of the DGM. Since we allow project crashing in the DGM but not in the CGM due to the complexities involved as discussed later, we split the projects which are crashed. All the characteristics of the split projects remain the same except for the durations. This improves the feasibility and allows us to use the solutions with good potential directly.
We later show that the quality of initial schedules has a negligible effect in our model. Once these schedules are generated, we iteratively solve the master and subproblems as explained next.

**5.6.1 The Master Problem (MP)**

The Master Problem (MP) is an MIP formulation where each decision variable, $x_r$, corresponds to a feasible solution. The variable $x_r$ is 1 if schedule (also called routes) $r$ is used in the solution, and is otherwise 0. The MP solves the global problem quickly if all of the feasible routes, $R$, are readily available. Our objective is to find a subset of $R$ which satisfies all of the MP constraints with a minimum penalty. However, the modularity of the set $R$ is exponential. This issue is addressed by considering only one set of feasible routes in the MP and calling it the Restricted Master Problem (RMP). We iteratively add more potential routes by solving several subproblems, which are called **pricing problems**.

Now we will show the mathematical formulation of the MP. We keep the definitions of symbols from previous models and define new ones as required.

**Index:**

- $r$: Route.

**Parameters:**

- $R$: Set of all feasible routes.
- $c_r$: Cost of route $r$.
- $\delta_{pr}$: 1 if project $p$ is included in route $r$; 0 otherwise.
- $t_{rk}$: 1 if crew-type of route $r$ is $k$; 0 otherwise.
- $p_{rw}$: Project on which route $r$ is working in week $w$.
- $s_p$: Subdivision in which project $p$ comes.
- $SW_{pr}$: Start week of project $p$ in route $r$.
- $EW_{pr}$: End week of project $p$ in route $r$.
- $A_{prw}$: 1 if project $p$ is active in route $r$ in week $w$. 

136
Adj_p: 1 if project p ∈ P^adj.

**Decision Variables:**

x_r: 1 if the route r is selected; 0 otherwise.

v^\text{curf}_w: Violation in at-most-curfew constraints in week w.

v^\text{subm}_{s1s2,w}: Violation in mutually exclusive constraints in week w.

v^\text{serv}_{uw}: Violation in service corridor constraints in week w for corridor u.

v^\text{prec}_{pq}: Violation in precedence constraints for pair pq.

**Objective:**

Minimize \sum_{r \in R} c_r x_r + \sum_{w \in W} c^\text{curf}_w v^\text{curf}_w + \sum_{s1, s2 \in S^{ME}} \sum_{w \in W} c^\text{subm}_{s1, s2, w} v^\text{subm}_{s1, s2, w} + \sum_{u \in U} \sum_{w \in W} c^\text{serv}_{uw} v^\text{serv}_{uw} + \sum_{pq \in P^{prec}} c^\text{prec}_{pq} v^\text{prec}_{pq}

**Constraints:**

\[ \sum_{r \in R} \delta_{pr} x_r = 1 \quad \forall p \in P \quad (5-22) \]

\[ \sum_{r \in R} t_{rk} x_r \leq \alpha_k \quad \forall k \in K \quad (5-23) \]

\[ \sum_{r \in R} \gamma_{prw} x_r I_{prw} = M_{swy} \quad \forall w \in W, s \in S \quad (5-24) \]

\[ \sum_{s \in S} y_{sw} = \mu + v^\text{curf}_w \quad \forall w \in W \quad (5-25) \]

\[ y_{s1w} + y_{s2w} = 1 + v^\text{subm}_{s1s2,w} \quad \forall (s_1, s_2) \in S^{ME}, w \in W \quad (5-26) \]

\[ \sum_{r \in R} A_{prw} x_r + \sum_{r \in R} A_{qrw} x_r = 1 + v^\text{prom}_{p1,p2} \quad \forall (p, q) \in P^{ME}, w \in W \quad (5-27) \]

\[ \sum_{s \in S} \lambda_{uw} y_{sw} = 1 + v^\text{serv}_{uw} \quad \forall u \in U, w \in W \quad (5-28) \]

\[ \sum_{r \in R} SW_{qr} x_r - \sum_{r \in R} SW_{pr} x_r = 1 + v^\text{prec}_{pq} \quad \forall (p, q) \in P^{prec} \quad (5-29) \]

\[ x_r \in \{0, 1\} \quad \forall r \in R \quad (5-30) \]

The objective function of the RMP is to minimize the cost of the selected routes and penalty occurred by constraint violations. The cost of a route r \{Project: p_1, p_2, \ldots, p_t\}
\[ c_r = \sum_i \text{dist}[p_i, p_{i+1}], \] where \( \text{dist}[p_i, p_{i+1}] \) is the distance between projects \( p_i \) and \( p_{i+1} \). We exclude the project’s pairs which are within the maximum distance limit from the costs. We assume that a crew always start from the source (\( p_1 \): source node) and finishes at the sink (\( p_t \): sink node). Constraints 5–22 ensure that each project is covered by exactly one route and Constraints 5–23 keep track of the number of crews employed. Constraints 5–24 and 5–25 force the number of absolute curfews within the given limit. Constraints 5–26 and 5–27 ensure the mutually exclusivity constraints at the subdivision and project levels, respectively. Constraints 5–28 keep a check on the number of service corridors under absolute curfew, and Constraints 5–29 maintain the precedence relations. Constraints 5–30 enforce the variables’ integrality. The values in parentheses are the dual values used in the pricing problems. We now discuss the Pricing Problem, which is solved repetitively to insert the potential candidate routes into the RMP.

5.6.2 The Pricing Problem

The Pricing Problem uses the concept of duality in linear programming (LP) and finds better candidate routes to include in the RMP. All routes sent from this stage should satisfy three constraints: (i) distance, (ii) time-window, and (iii) precedence-relations. In each iteration, we solve four elementary shortest-path problems with resource constraints (ESPPRC) for each crew-type. The underlying network for each ESPPRC is constructed as follows: we create the nodes \( \{0, 1, \ldots, N, N+1\} \), where 0 and \( N+1 \) are source and sink nodes, respectively. The arc costs are calculated based on the current dual values of the RMP constraints (see the mathematical formulation in Section 5.6.1). We need to revise the arc costs of the pricing problems each time we solve the RMP and seek new candidates to enter. We also need to dynamically calculate the arc costs, because the start-times of unscheduled projects are unknown.
We only show the formula used to revise these arc costs, but the details can be found in most literature on column generation [10]. For a partial path that ends with node $p$, the modified arc cost $\hat{c}_{pq}$ for the crew-type $k$ is given as:

$$c_{pq} - \pi_2^2 - \sum_{w=W_p}^{E_p} \gamma^p \pi_{w}^4 - \sum_{w=W_p}^{E_p} \text{Adj}_p \gamma^p \pi_{(p,q)}^7 - \text{Prec}_p SW_p \pi_{pq}^9 \text{ if } p \in P \text{ and } q \in P \cup N + 1$$

$$c_{pq} - \pi_k^3 \text{ if } p \in 0 \text{ and } q \in P$$

Based on our observations in the computational experiments, one of the most critical issues with creating a time-efficient algorithm is solving the sub-problem efficiently. The sub-problem, which is an ESPPRC, is strongly NP-Hard [18]. Instead of solving it using an MIP, we use dynamic programming and obtain a set of candidate routes to be sent, not only the optimal one. This reduces the number of iterations and therefore improves the running time for our case. The basic framework of the dynamic algorithm is shown below in Figure 5-3. In Figure 5-4, we summarize the algorithmic flow of the column-generation model (CGM) implemented for the CPP.

### 5.6.3 Computational Analysis

We tested the CGM on the same real-life instances and found that it was very effective in terms of solution quality as well as the running time. We used the output of the DGM as part of the initial routes. The CGM generated the solutions of either a similar or slightly better quality than that of the DGM within 20 minutes of running time. Although we expected to see a major improvement, the solutions were not significantly improved when the output of the DGM was supplied. However, the solutions definitely were better than the TSNF in time as well as in quality. After some analysis, we realized that the newly generated potential routes were not very effective, because most of the constraints which affect the solution qualities are modeled in the MP but not in the subproblems. The second reason is that we didn’t allow project crashing, for it would have increased the complexity of the repeatedly solved subproblems. The details of our computational results are discussed in Section 5.8. The combined results of the CGM and the DGM were good, but the number of distance constraints violations in
**Basic Dynamic Algorithm Framework**

Inputs: Updated arc costs, time window of each node, and precedence matrix.

Output: A set of elementary paths from node 0 to node N+1 that have negative reduced costs and do not violate any capacity constraints.

**Step 1: Initialization**
- Get the updated arc costs associated with the last RMP solved.
- Create Label (\(\{0, 0\}\)).
- Initialize a set of sets that stores the elements of each distinct path found.
- Initialize the predecessor set.

**Step 2:** \(\textbf{WHILE } u \neq N+1 \textbf{ DO}\)

\(\{u, tu\} = \text{BestLabel (unprocessed labels)}\)

\(\text{IF } u < N+1 \text{ THEN}\)
- \(\text{FOR each } v \in P \cup \{N + 1\} \text{ DO}\)
  - \text{Feasibility Check:}\n    - \(\text{IF } u \neq v \text{ AND } \hat{c}_{uv} \neq \infty \text{ AND } tu + tv = |W| \text{ AND } v \neq \text{preds}[[u, tu]] \text{ THEN}\)
      \(tv = tu + tv\)
  - \text{Improvement Check:}\n    - \(\text{IF } \text{Cost}([u, tu]) + \hat{c}_{uv} < \text{Cost}([v, tv]) \text{ THEN}\)
      - \(\text{isCreatedLabel} = \text{DominanceRule()}\)
      - \(\text{IF isCreatedLabel = true THEN}\)
        - \(\text{CreateLabel}([v, tv])\)
        - \(\text{preds}[[v, tv]] = \text{preds}[[u, tu]] \cup \{v\}\)
      - \(\text{IF } v = N + 1 \text{ AND } \text{Cost}([u, tu]) + \hat{c}_{uv} < 0 \text{ THEN}\)
        - Form a set from the nodes of the distinct path.
        - \(\text{IF } \text{The node set of the path is distinct THEN}\)
          - Keep the order of the node set.
        - \(\text{ELSE IF } \text{The node set already exists THEN}\)
          - Update the order of the node set
    - \(\text{Remove label } \{u, tu\} \text{ from the set of unprocessed labels.}\)

**Step 3:** \(\text{FOR each distinct path sets DO}\)

- Prepare the routes found for the MP.

Figure 5-3. Dynamic programming algorithm for ESPPRC.

these solutions was very high. As discussed earlier, we could not enter even the partial distance constraints in the second phase of the DGM. The third phase is restrictive once the project schedules are fixed, so, we also could not improve the results significantly.

This encouraged us to develop a decomposition-based approach in which we could use the distance constraints in a more efficient manner without strongly compromising other constraints' violations. We discuss the approach, which was found to be both practical and satisfactory, in the next section.

140
Steps of the Column-Generation Procedure
Inputs: Distance between projects, number of crews of each type, precedence matrix, and time windows for the projects.
Output: Schedule for each crew.

Step 1: Initialization.
- Get the inputs.
- Create departure depot (D) and arrival depot (A) sets.
- Initialize routes: Construct multi-node routes using heuristics.
- Prepare initial set of routes for the MP.

Step 2: Construct the MP.

Step 3: Solve the RMP with current set of feasible routes.
- Update the lower bound.
- Get the dual values for each constraint of the RMP.
- Modify the arc costs using the dual values.
  FOR each crew type DO
    - Solve subproblems with modified costs.
    IF ReducedCost < 0 for any new path r, go to Step 4.
    ELSE go to Step 5.

Step 4: Add the new promising columns to the RMP and go to Step 3.

Step 5: Optimal solution for the original relaxed master problem has been found.

Figure 5-4. Algorithm framework of the column-generation procedure.

5.7 Decomposition-Based Duty Generation Model

The CPP with all of its attributes, such as project crashing, distance limits, service corridors, etc., is very complex. We tried three holistic models to solve it, but only two were successful with some limitations. All of the models had distance limits, which were the hard constraints that reduced the performance. The main concern became figuring out how to incorporate the distance constraints in the main model and thus improve the solution quality. This led us to decomposing the problem into several parts and then solving each part sequentially. Our first idea was to repetitively solve the DGM by setting the distance constraints for five weeks in each iteration. The algorithm for the repetitive DGM has following steps:

Step 1: Set $w_1 = 0$, $w_2 = 5$. Step 2: Solve the DGM in Phases 1 and 2 with the distance constraints only for weeks $w_2$ to $w_2$.

Step 3: Fix the schedule from week 0 to $w_2$. Update the duty matrix.
Step 4: If \( w_2 = W \), go to Step 5 and set \( w_1 = w_2 \). If \( w_2 < 40 \), \( w_2 = w_2 + 5 \), else \( w_2 = W \). Go to Step 2.

Step 5: Solve Phase 3 of the DGM and generate the schedules of each crew.

The repetitive DGM had mixed success as it improved the solution quality for initial set of weeks but deteriorated for the rest. Overall, we found that the total number of violations was reduced but not significantly. The marginal success of the repetitive DGM caused us to try breaking the problem into three parts and then use all of the partial distance constraints for each of them. We call this approach our decomposition-based model, and the steps are given below:

Step 1: Solve the Phase 1 and Phase 2 of the DGM without any distance constraints. We call the project schedule obtained here a *Rough-Schedule*.

Step 2: Divide the projects into three parts: (a) 1-15 weeks, (b) 15-30 weeks, and (c) 30-50 weeks. Each part contains the projects that were active in the corresponding time span of the Rough-Schedule. Set \( i = 1 \).

Step 3: Solve \( i^{th} \) part using the DGM results from Phases 1 and 2 and all partial distance constraints.

Step 4: Fix the solution of \( i^{th} \) part and supply the boundary schedule (15th week projects). Set \( i = i + 1 \). If \( i = 4 \), go to Step 5, else go to Step 3.

Step 5: Solve Phase 3 of the DGM and generate the schedules of each crew.

There are several implementation-level details for the decomposition-based algorithm. In parts (a) and (b), we forced the crew conservation constraints for each week; while in part (c) we relaxed it after the 45th week. For parts (a) and (b), we may either force the projects to be completed within the same part or relax them for later scheduling. We implemented both strategies and found better results in the case where we forced them to be completed in the same part. When using a relaxed strategy, the parts (a) and (b) generated very good results but, the bad projects (projects that were less time-flexible) accumulated in the last part, (c), which caused high number of
violations. Another main decision issue was distributing the total available running time (eight hours) among different parts. Our first step was to generate the Rough-Schedule, which helps in partitioning the projects. Generating good-quality schedules requires excellent project partitioning. Therefore, we allotted four hours for it and divided the remaining four hours equally among the three parts. If any part reached optimality within the allotted time, the remaining time was added to the next part. These strategies were effective for our problems and generated very sound solutions, as we discuss in the next section.

5.8 Computational Results

We tested our models on the real-life instances of a major North American railroad company (XYZ). These instances were for preparing the annual schedules of the years 2007 and 2008. The number of projects and hence other inputs were of the same order for both years. Table 5-1 shows the size of inputs in detail. Generally, the projects are equally divided between rail and tie types. Each service corridor consists of anywhere between four and ten subdivisions. Around 500 subdivision pairs are mutually exclusive, and each subdivision lies in three service corridors on average. A project is within the maximum distance limits of around 15-25 other projects. Railroads allow at the most 15 absolute curfews in each week. We assume that each violation causes the same penalty, although it may differ among railroads based upon their priorities.

Table 5-1. Details of real-life instances

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Year 2007</th>
<th>Year 2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Projects (Rail/TS)</td>
<td>254(139/115)</td>
<td>265(150/115)</td>
</tr>
<tr>
<td>Rail Crews (Small/Large)</td>
<td>10(9/1)</td>
<td>10(7/3)</td>
</tr>
<tr>
<td>Tie Crews (Small/Large)</td>
<td>9(2/7)</td>
<td>9(2/7)</td>
</tr>
<tr>
<td>Number of Weeks</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Number of Service Corridors</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Number of Subdivisions</td>
<td>282</td>
<td>282</td>
</tr>
</tbody>
</table>

We have implemented all of our models in C++ using CPLEX 11.2 on a 2.39 GHz PC with 1.99 GB of RAM. The results obtained by these algorithms and those of the XYZ railroad are compared in Table 5-2 and Table 5-3 for the years 2007 and 2008,
respectively. We also have included the best solutions available for the same instances as given in the paper by Bog et al. [15]. They developed several heuristics to solve the problem and compared them with railroad-implemented schedules. We have improved those results and set new benchmarks. As stated earlier, the TSNF results as stated were not very good, even though they are marginally better than XYZ’s schedules. We were unable to get feasible solutions by solving the global TSNF, so we decomposed it into rail and tie projects. We solved each partition separately but sequentially, starting with the rail projects. Once we had the schedules of the rail projects, all parameters such as the remaining absolute curfew allowed per weeks were updated for the tie projects. For each partition, we set a time-termination criteria of four hours. Solving the problem in parts was not effective with respect to the solution quality. But we could improve it by solving the partitioned problems repetitively by updating the parameters from each previous iteration; however, this was not a practical approach given the time-limit of eight hours.

The DGM is very effective if the restriction on maximum inter-project distance travelled is relaxed. The main attribute which makes the model effective is the inclusion of project crashing in the model. Project crashing improves the feasibility significantly by allowing a project to be done by two different crews and thus decreasing the chances of exceeding the maximum curfew limit, violating service corridor restrictions, and so on. The main limitation of the DGM is its inability to effectively handle distance constraints. Even adding partial distance constraints in the model makes the size of the model too large to solve within the time limit using the given hardware configurations. We therefore solve them without distance constraints and pass these schedules as initial solutions to the CGM. We see that the CGM improves the solution marginally for the year 2007 but doesn’t improve it for the year 2008. The advantage of CGM is that it distributes the distance violations of the DGM over all other violations and generates a solution with better uniformity. It may be a more acceptable solution to the railroads
for which the distance limit violations are very costly. The CGM could not improve the schedules, because most of the main constraints are handled in the Master Problem. The subproblem only deals with time-window constraints, distance constraints, and some precedence constraints. Hence, the routes generated in subproblems don’t have very good qualities with respect to other constraints. Another reason for the limited improvement is the inability of the CGM approach to crash the projects internally. The projects that are crashed by the DGM are supplied as two separate projects in the CGM, hence, the crashed projects are pre-decided.

Table 5-2. Improvement in number of violations for 2007

<table>
<thead>
<tr>
<th>Parameters</th>
<th>XYZ</th>
<th>Bog et al. [15]</th>
<th>TSNF</th>
<th>DGM</th>
<th>CGM</th>
<th>Decomposed DGM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute Curfew</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>Mutually Exclusive</td>
<td>29</td>
<td>1</td>
<td>28</td>
<td>0</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td>Service Corridor</td>
<td>21</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Time Window</td>
<td>154</td>
<td>78</td>
<td>107</td>
<td>24</td>
<td>32</td>
<td>28</td>
</tr>
<tr>
<td>Distance</td>
<td>60</td>
<td>42</td>
<td>93</td>
<td>127</td>
<td>11</td>
<td>29</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>266</td>
<td>132</td>
<td>245</td>
<td>151</td>
<td>148</td>
<td>57</td>
</tr>
</tbody>
</table>

Table 5-3. Improvement in number of violations for 2007

<table>
<thead>
<tr>
<th>Parameters</th>
<th>XYZ</th>
<th>Bog et al. [15]</th>
<th>TSNF</th>
<th>DGM</th>
<th>CGM</th>
<th>Decomposed DGM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute Curfew</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>Mutually Exclusive</td>
<td>43</td>
<td>6</td>
<td>25</td>
<td>0</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>Service Corridor</td>
<td>52</td>
<td>26</td>
<td>27</td>
<td>0</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>Time Window</td>
<td>7</td>
<td>0</td>
<td>21</td>
<td>7</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Distance</td>
<td>30</td>
<td>22</td>
<td>51</td>
<td>96</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>132</td>
<td>54</td>
<td>127</td>
<td>103</td>
<td>103</td>
<td>21</td>
</tr>
</tbody>
</table>

The solutions obtained by the Decomposed DGM are the most effective from among all of the models. It decreases the number of violations of each constraint and hence the number of total violations significantly. The number of total violations is almost 25% of the implemented schedules for the year 2007, while it is 10% for the year 2008. These solutions also are improved compared to the solution quality obtained by Bog et al. (2009), but the time taken to get these solutions is higher. However, the higher running time is justified as the schedules are prepared only once per year and revised sporadically.
To check the effectiveness of the approach, we compared schedules generated by all of the models as well as those implemented by the railroad in corresponding years. We also tried comparing it with solutions obtained from exact algorithms modeling all constraints. It was not possible to find the optimal schedules for the real-life instances due to the large size of the problems and high complexity of the current computational capabilities. So, we created some small instances. These instances have the same network characteristics as real ones, such as the ratio of the number of weeks available and the number of weeks required to complete the given set of projects, the ratio of mutually exclusive project pairs to total pairs, and so on. We created 10 instances with a five-week planning horizon and 10 instances with a 10-week planning horizon. We then tried to solve each of the instances using the exact MIP formulation described in the paper by Bog et al. [15]. Their formulation captures all issues of the problem, but it is intractable even for modest-sized instances. We solved the exact model using CPLEX 11.2 on a 2.5 GHz PC with 2 GB of memory. We set the termination criteria based on a maximum available time of eight hours, even for smaller instances. It generated optimal solutions for six out of ten five-week instances, while there were optimality gaps of 3% to 8% for the rest [15]. The results obtained by Bog et al. [15], TSNF, DGM, CGM, and DGM* (explained next) for five-week instances are shown in Table 5-4. The TSNF generates optimal solutions for five-week instances in around six minutes, but project crashing is not included. The quality of the TSNF is not acceptable even for these small instances as the gap varies from 2% to 15%, which clearly demonstrates the importance of project crashing. We tested the DGM by putting all partial distance constraints (equivalent to the decomposed model for smaller instances) in these instances. It generated optimal solutions for eight out of ten instances. In the remaining two instances, the number of violations was only one more than the optimal solutions. We used these schedules as the initial solutions of the CGM, which improved the solution of one of these two. We also tested the effect of using the pair-wise duties’ distance
constraints (for each pair of duties) along with individual duty-distance constraints as discussed in Section 5.5. The results are shown in Table 5-4 in column DGM*. It converged to the optimal solution for nine out of ten instances. These results show that the partial-distance constraints are very effective in these real-life instances that have projects scattered over a large geographical region.

We also tested our algorithms on ten-week instances. The exact algorithms could not generate even one feasible solution for any of these instances, again emphasizing the very high complexity of the problem. The results of other models show a similar behavior, with the DGM generating the best results consistently.

Table 5-4. Comparison of algorithms: optimality gap (small instances)

<table>
<thead>
<tr>
<th>Instances</th>
<th>Bog et al. [15]</th>
<th>TSNF</th>
<th>DGM</th>
<th>CGM</th>
<th>DGM*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00%</td>
<td>4.76%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>2</td>
<td>5.77%</td>
<td>8.77%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>3</td>
<td>3.70%</td>
<td>6.90%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>4</td>
<td>11.54%</td>
<td>7.14%</td>
<td>3.85%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>5</td>
<td>1.96%</td>
<td>13.56%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>6</td>
<td>0.00%</td>
<td>2.13%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>7</td>
<td>5.13%</td>
<td>9.30%</td>
<td>2.56%</td>
<td>2.56%</td>
<td>2.56%</td>
</tr>
<tr>
<td>8</td>
<td>4.55%</td>
<td>10.20%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>9</td>
<td>8.51%</td>
<td>7.84%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>10</td>
<td>2.86%</td>
<td>14.63%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

The running time of our models for these instances was far better than those of the exact model, because all of the solutions were obtained within one minute. We also attempted to test the 10-week instances, but the exact algorithms simply could not generate any solution within 8 hours, while our Decomposed-DGM model could solve them within two minutes. These tests show the quality and practicality of the model for handling real-life scenarios.

5.9 Conclusions

In this paper, we consider a Curfew Planning Problem (CPP) that causes a significant part of the total maintenance costs of the railway network of railroads, and hence even a little improvement in the planning will result in very significant savings.
Considerations of all the issues of real-life applications made the problem particularly complex and non-linear in nature. We have proposed three exact approaches and one decomposition-based heuristic to solve the problem. The time-space network formulation (TSNF) was not very successful under the current scope of the problem, while the other two approaches (DGM, and CGM) had mixed successes. The decomposition-based heuristics was the stand-out among all of the other models as it generated the best results within the allotted time interval. All of our experiments were based on recent real-life data obtained from a large North American Railroad company. We were able to improve the schedules employed by the company significantly – by up to 70%. The main bottlenecks for all of the models were the distance constraints, and we tried to handle them in several ways. The most successful solution involved entering partial constraints after dividing the projects into several groups that were formed after solving a global problem. Future research can be performed in the direction of utilizing these constraints in improved and even more efficient ways.
CHAPTER 6
GENERAL CONCLUSION AND FUTURE RESEARCH

In this dissertation we have discussed three large scale scheduling problems occurring mostly in the railroads. The load scheduling problem, arising at an intermodal railroad terminal, is to assign the containers and trailers on the given set of railcars to maximize train utilization and aerodynamic efficiency. The solution to the problem must also satisfy several operational and regulatory requirements. We formulate the LSP as an integer program on an underlying network and solve it to optimality using CPLEX. The run-time of the CPLEX optimizer increases exponentially with the network size, and it fails to solve the modest size problems in reasonable time. To efficiently solve these real-life instances, we propose two multi-exchange neighborhood search algorithms, and one hybrid approach. Our empirical studies demonstrate that these algorithms are able to solve the problems of modest size in reasonable time. The hybrid algorithm generates the best results among all for all test instances within a few minutes.

The subset disjoint minimum cost cycle problem is to find the least cost cycle in a network which contains at most one node from any subset in which nodes are divided. These problems are encountered mainly as the subproblems of other highly combinatorial problems discussed several exact and heuristic algorithms. The exact algorithms were inspired by pre-existing dynamic programming algorithms for resource constrained shortest path and all pair shortest path algorithms. We propose the more generalized version of these algorithms along with several strategies to reduce the network and size and thereby reduce their running time. Since, these problems are used as a sub-problems of more complex problems and solved repetitively, a running time of more than a second may not be effective. We developed several heuristic approaches each controlling a specific parameter of exact algorithms. Among all these heuristics, limited unprocessed labels (LUL) implemented on APPull algorithms performed better than all other heuristic combinations for SDMCC as well as SDNCC problems. These
arguments were supported by extensive computational tests performed on wide variety of instances. We can also conclude that if the network contains a negative cycle, then to find minimum cycles, the algorithms specific for SDNCC problems should be used as optimal cycle must be negative.

The Location-Routing Problem (LRP) is a combination of two classical optimization problems: facility location and vehicle routing. We first introduce a formal MILP model where we have multiple capacitated vehicles and facilities. Since, a direct application of optimization software becomes ineffective even for a small size problem due to its immense complexity, we apply a column generation algorithm tailored to the solution of this kind of LRP. This research is the first study that proposes a column generation algorithm for the solution of the general LRP that has capacitated multiple facilities and vehicles. We decompose the LRP into two subproblems. The master problem is a set partitioning problem with side constraints and the subproblem is an elementary shortest path problem with resource constraints. We apply column generation technique to solve the relaxed master problem (RMP) and a new dynamic programming algorithm been designed in order to generate the columns in the subproblems. The gaps obtained are tightened since we solve an elementary shortest path problem eliminating all k-cycles. The column generation algorithm has been successful in solving instances up to 10 candidate facility locations and 100 customers. For the larger instances the number of labels created at each solution of the subproblem has been very large and the running times to find a set of good routes increased. Therefore, the algorithm ran till the time limit for 5 of the 18 instances. Further work for the early elimination of infeasible or unnecessary labels is required to solve larger instances efficiently. For instance, adding a more sophisticated dominance criterion should improve the efficiency of the dynamic programming procedure hence affect the overall performance of the column generation algorithm. Future work should also focus on adding cutting planes to the master problem to allow larger instances to be solved.
The Curfew Planning Problem (CPP) is encountered by railroads to maintain their railway tracks. The CPP is to design an optimal annual timetable to complete a given set of repairs and replacement jobs (rail-work and tie-work) on the railway tracks for a set of crews specialized in rail-work (rail-crew) or tie-work (tie-crew). We develop the maintenance-schedule for each crew such that the disruptions in train routes due to station-curfews are minimized. We give four solution approaches for the CPP: (i) time-space network, (ii) duty-Generation, (iii) column-generation, (iv) decomposition-based duty generation model, and (v) optimization-based heuristics. We solve each formulation using CPLEX and present the computational results based on a real life (a major North American Railroad) as well as simulated instances. The results generated by these algorithms are show significant improvement over currently implemented solutions.
REFERENCES


BIOGRAPHICAL SKETCH

Ashish Nemani was born in 1984, in Bihar, India. He received his bachelor's degree in industrial engineering at the Indian Institute of Technology, Kharagpur, India in 2005. He received his master's degree in industrial engineering at the University of Florida, Gainesville, Florida, in 2007. He has been a doctoral student in the Department of Industrial and Systems Engineering at the University of Florida since August 2005.