APPROXIMATION ALGORITHMS FOR POSTMAN PROBLEMS

APPROVED BY SUPERVISORY COMMITTEE:

Supervisor: ________________________________

APPROXIMATION ALGORITHMS FOR POSTMAN PROBLEMS

by

Jeyakesavan Veerasamy, B.E., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May 1999

This dissertation is gratefully dedicated to

Veerasamy, my father,

Seeniammal, my mother, and

Balakrishnan, my brother,

for their support, their guidance, and their faith.

ACKNOWLEDGEMENTS

Part-time Ph.D. students come up with their own flavor of problems, and require a lot more patience and support from their advisors. Dr. Balaji has helped me in all the ways possibly he can, without giving up hope in me in the first few years in which I struggled to get a footing in my research work.

Many thanks to committee members Dr. S. Venkatesan, Dr. R. Chandrasekaran, and Dr. Klaus Truemper for agreeing to be in the committee and providing suggestions for new research ideas. Additionally, special thanks to Dr. Venkatesan for supporting me as research assistant when I was full-time Masters student, and for encouraging me to do part-time Ph.D.

I would like to thank my dedicated parents Veerasamy and Seeniammal for unconditionally supporting me emotionally and financially in all of my endeavors. I would like to thank my wife Priya for her patience and understanding when I was busy preparing this dissertation.

I would like to thank the faculty of the Computer Science Department at the University of Texas at Dallas who helped me to broaden my knowledge in Computer Science.

I would like to thank my previous and current employers for their understanding and financial support towards my education and research. Particularly

I would like to thank Kevin Swank, Chuck Bradshaw, and Barbara Ocone at Northern Telecom Inc. and Arshad Mirza and Bruce Brady at Samsung Telecommunications America Inc.

Finally, I would like to thank my friends for providing a lot of encouragement along the way, especially Chandra and Uma, Sridhar and Srimathi, Sudhir and Sushma, Prabhakar and Vanitha, and Balaji and Durga for helping me to keep up my emotional and physical health.

March 1999.

APPROXIMATION ALGORITHMS FOR POSTMAN PROBLEMS

Publication No. ________________

Jeyakesavan Veerasamy, Ph.D.
The University of Texas at Dallas, 1999

Supervising Professor: Balaji Raghavachari

Problems of finding paths and tours in graphs is of fundamental importance and they have many practical applications. The traveling salesman problem is to find a minimum-cost tour of a given graph, covering all nodes exactly once. The Euler tour problem is to find a tour of a graph covering all edges exactly once. It is known that an Euler tour exists if and only if each node of the graph is of even degree. The Postman problem is to find a minimum-cost tour of a graph covering all edges at least once. In other words, it is the optimization version of the Euler tour problem, and the goal is to add additional edges of minimum cost such that the resulting graph is Eulerian.

The postman problem is similar to the well-known traveling salesman problem at first glance, but they are quite different in terms of complexity of the problems. While the traveling salesman problem is NP-hard, the Postman problem can be solved in polynomial time if the input graph contains only undirected

edges or if it contains only directed edges. If the input graph is mixed, i.e., if it contains both undirected and directed edges, then the problem is NP-hard, and this version of the problem is known as the mixed postman problem. Similarly, if the input graph is undirected, but the edge traversal costs are asymmetric in its two directions, then too, the problem is NP-hard. This problem is called the windy postman problem. Prior to our work, the best known approximation ratio for the mixed postman problem and the windy postman problem were $5/3$ and 2 respectively. We study the properties of feasible solutions to these problems, derive new lower bounds for the costs of optimal solutions, and design approximation algorithms with approximation ratio of $3/2$, for both problems.

Additionally, there are other interesting variations of the postman problem. One variation is to limit the mixed postman tour to duplicate only a subset of undirected edges and directed edges. We present an optimal algorithm for this restricted mixed postman problem for even degree input graphs. We present a 3/2-approximation algorithm for general graphs, when the duplication is restricted to all undirected edges. Another interesting variation of the postman problem is to restrict the turns allowed at each node. We prove that the postman problem with restricted turns is NP-complete. Several other variations of the postman problem are also discussed.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Graph theory is one of the most important areas of Computer Science. A wide variety of practical problems can be modeled using graph theory. Application areas include Operations research, Electrical engineering, Industrial engineering, Science, Civil engineering, Architecture, Geography, Social sciences, Economics, puzzles, and games. Refer to the books by Bondy and Murty [3], Even [16], and Foulds [19] for detailed information on various applications.

Several problems of practical importance have been proved to be NP-hard, and these problems are not known to have polynomial-time algorithms. Therefore, researchers use heuristics to solve them approximately. A key performance indicator of an approximation algorithm is its approximation ratio, which is the maximum ratio between the cost of the solution computed by the algorithm and the cost of the optimal solution over all input instances.

1.1 Traveling salesman problem

Problems of finding paths and tours on graphs have been widely studied due to their fundamental importance and practical applicability. The traveling salesman problem (TSP) is a widely studied and well publicized problem. Given a graph, the goal is to find a minimum cost tour visiting each node exactly once. TSP has a lot of practical applications in vehicle routing, job sequencing, computer wiring, cutting wallpaper, etc., but it is known to be NP-hard, i.e., there is no known polynomial-time algorithm for this problem. In fact, there is not even a good approximation algorithm if the edge-costs are unrestricted. Once the problem constraint is relaxed from visiting each node "exactly once" to "at least once", there exists an approximation algorithm due to Christofides [5] which guarantees a solution whose cost is at most 1.5 times the cost of an optimal solution for a given instance of the problem. Refer to the book by Lawler, Lenstra, Rinnooy Kan, and Shmoys [31] for a detailed discussion and references on the traveling salesman problem.

1.2 Euler tour problem

Another important traversal problem is to find a tour in a given graph that traverses each edge exactly once. Such a tour is called an Euler tour. The Königsberg bridge problem (Figures 1.1 and 1.2), the first problem to be studied in graph theory, was posed in the 18th century: the puzzle involved starting on any one of the pieces of land and finding a tour that crossed each of the seven bridges exactly once and finally returning to the starting point. In 1736, Swiss mathematician L. Euler proved it to be impossible, and gave necessary and sufficient conditions for the existence of a tour in an arbitrary system of interconnected bridges [14, 15]. He proved that such a tour is possible only if every point is incident with an even number of lines. Graphs in which such tours exist are called Eulerian graphs, or simply Eulerian.

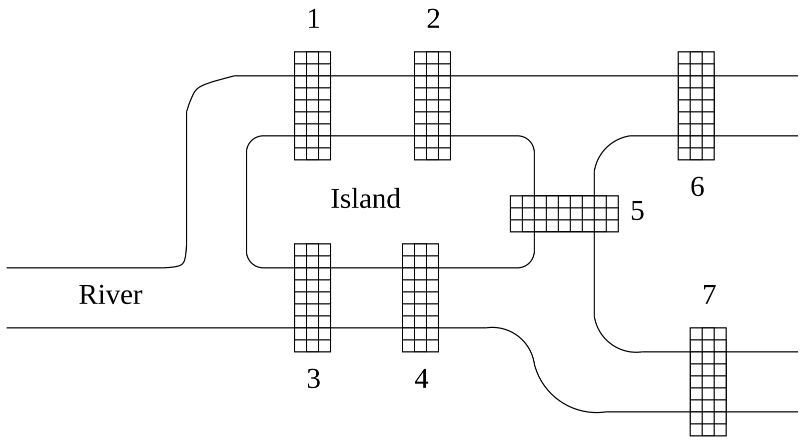


Figure 1.1: Königsberg bridge problem

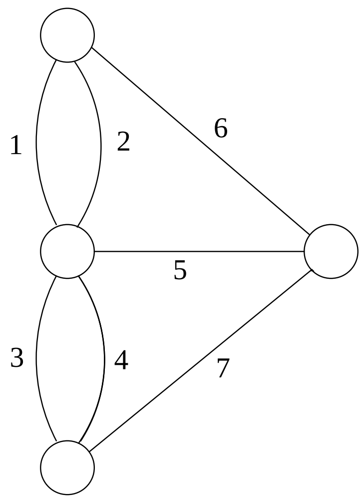


Figure 1.2: Graph underlying the Königsberg bridge problem

1.3 Chinese postman problem

Since an Euler tour does not exist in all graphs, a natural question is to determine a minimum-length tour covering each segment at least once. This problem is known as the Chinese postman problem (CPP) since it was first posed by Meigu Guan, a Chinese mathematician [25]. His problem statement is as follows:

> "A mailman has to cover his assigned segment before returning to post office. The problem is to find the shortest walking distance for the mailman."

A generalization of the postman problem, known as the rural postman problem (RPP), is to determine a least-cost traversal of a specified subset of edges in the graph.

Edge traversal problems arise in a number of practical contexts, such as mail delivery, garbage collection, snow removal, and school-bus routing. Billions of dollars are spent each year by governments and private enterprises on these operations. A sizable portion of this money is wasted because of poor planning. Computer scientists, mathematicians, and operations researchers have for a long time studied the structure of these problems and proposed workable solutions, sometimes yielding sizable savings in cost. Refer to survey articles by Eiselt, Gendreau, and Laporte [12, 13] for more details.

1.4 Mixed postman problem

If the input graph to the postman problem contains only undirected edges or only directed edges, there exist polynomial-time algorithms to solve them optimally. However, if the input graph contains both undirected edges and directed edges, then the problem is NP-hard, and no polynomial-time algorithm is known. The corresponding problem is called the mixed postman problem (MPP). Numerous articles have appeared in the literature over the past three decades about the mixed postman problem. Edmonds and Johnson [11], and Christofides [6] presented the first approximation algorithms for this problem. Frederickson [20] showed that the algorithm of [11] finds a tour whose length is at most twice the length of an optimal tour (i.e., approximation ratio of 2). He also presented a mixed strategy algorithm, which used the solutions output by two different heuristics, and then selected the shorter of the two tours. He proved that the approximation ratio of the mixed strategy algorithm is 5/3.

In this dissertation, we present an improved approximation algorithm for MPP with an approximation ratio of 3/2. We study the properties of feasible solutions to MPP, and derive a new lower bound on the cost of an optimal solution. Our algorithm uses a subtle modification of Frederickson's algorithm, and the improved performance ratio is derived from the new lower bound.

1.5 Windy postman problem

In several practical scenarios, the windy postman problem (WPP), a generalization of CPP and MPP is more appropriate. This problem is known as the windy postman problem, because, as implied by the name, due to wind (or slope, or some other factor), the cost of traversing an edge in one direction may be cheaper than traversing the edge in the opposite direction. The underlying graph is an undirected graph, so the edges can be traversed in either direction, but with different costs. The goal is to find a minimum-cost tour traversing each edge at least once. This problem is also NP-hard [4, 26]. Undirected CPP, directed CPP, and MPP all are special cases of WPP.

Minieka [35] introduced the Windy postman problem in 1979. Guan [26] proved that the problem can be solved in polynomial time if both orientations of every cycle of the graph results in same cost. Win [51] proved that WPP is solvable in polynomial time if the input graph is Eulerian, i.e., if it is connected and has an even number of edges incident at every node. Win also presented a Linear Program (LP) to orient the edges and form directed tours. He proved that the LP gives an integer solution for Eulerian graphs, and half-integral solution for general graphs. For general graphs, Win suggested to add a minimum-cost perfect matching of odd-degree nodes based on the average costs of the edges to make the graph Eulerian and then obtain an integral solution to the LP. Since a minimum-cost T-join duplicates each edge at most once, Win's algorithm has an approximation ratio of 2.

We present a new approximation algorithm for WPP by combining the LP formulation for WPP and Frederickson's mixed strategy [20]. We achieve an approximation ratio of 3/2 for WPP by observing critical properties of intermediate solutions and proving an improved lower bound for the optimal solution.

Significance of our results: In many instances, problems on graphs with asymmetric costs seem much harder than when the cost function is symmetric. Examples include the traveling salesman problem [31] and Steiner tree problem among many others. In metric spaces, the asymmetric versions of these problems seem to differ dramatically from their symmetric counterparts in terms of their approximability. The main obstacle seems to arise from the fact that given a cycle, the cost of traversing the edges of the cycle in one direction could be arbitrarily large compared to the cost of traversing the cycle in the opposite direction. Prior to our work, the best known approximation ratio for the symmetric and asymmetric versions of the postman problem were 5/3 and 2 respectively. We close this gap by developing approximation algorithms for both versions of the postman problem with approximation ratios of 3/2.

1.6 Overview

The rest of the dissertation is organized as follows. In Chapter 2, common definitions and fundamental concepts related to graph theory are presented. In Chapter 3, we survey the algorithms for the undirected postman problem, the directed postman problem, and the mixed postman problem (MPP), and present a new approximation algorithm for MPP. In Chapter 4, we analyze Win's result for the windy postman problem and present our new approximation algorithm. Postman problems offer a rich collection of interesting variations. We discuss three variations in detail. In Chapter 5, we study MPP with duplication restricted to only a specified subset edges, and present an approximation algorithm for the case when duplication is restricted to undirected edges only. In Chapter 6, we study the complexity of CPP in which there is a cost associated with each connection at the nodes, and the goal is to minimize the total traversal cost which includes the edge-costs and the connection-costs. In Chapter 7, we review the rural postman problem in which the postman is required to visit only a specified subset of edges at least once. In Chapter 8, we review other variants of CPP known in the literature. We summarize the results of this dissertation and propose future work in Chapter 9.

Part of the results presented in this dissertation appear in preliminary form in [41, 42, 43].

CHAPTER 2

PRELIMINARIES

2.1 Definitions

A *graph* $G = (V, E)$ is an ordered pair, where V is a finite, nonempty set of elements termed *vertices* or *nodes*, and E is a set of unordered pairs of distinct vertices of V. Each element $(u, v) \in E$ is called an edge and joins the vertices u and v. We may also denote such edge as e_{uv}. When an edge is allowed to occur multiple times in E, such a graph is called a *multigraph*. However, we will not distinguish between graphs and multigraphs in this dissertation. A graph is a *connected* graph if there exists a path connecting every pair of nodes in the graph.

Sometimes edges may have an orientation or direction. If all edges are oriented, such a graph is called a *directed graph*, or *digraph* for short. A *digraph* is defined to be an ordered pair (V, A), where V is a finite, nonempty set of nodes, and A is a set of ordered pair of distinct elements of V. We call the elements of A directed edges or *arcs*. A digraph is *strongly connected*, if there exists a directed path from each node to every other node in the graph, going along the direction of arcs. A *mixed graph* $G = (V, E, A)$ consists of a set of vertices V, a multi-set of edges E, and a multi-set of arcs A.

A graph is termed *weighted* if there exists a function C which associates a rational number to each edge or arc called the *weight* or *cost* of that edge or arc. We extend the definition of C to a multi-set of arcs and edges by taking the sum of the costs of the corresponding edges and arcs.

A *cut* is a partition of the vertex set V into sets S and $V - S$. It is called *nontrivial* if neither side is empty. An edge *crosses* the cut if it connects a vertex in S to a vertex in $V - S$. Let $outdegree(v)$ be the number of outgoing arcs from v. Similarly, $indegree(v)$ is the number of incoming arcs into v. Let $degree(v)$ be the total number of edges and arcs incident to v. We say that v has even degree if $degree(v)$ is even. A graph has even degree if all its vertices have even degree. Let $surplus(v) = indegree(v) - outdegree(v)$. If $surplus(v)$ is negative, we may call it as a *deficit*. The definition of surplus can be extended to sets of vertices S, by taking the difference between outgoing and incoming edges that cross the cut $(S, V - S)$.

2.2 Matchings and T-joins

Given an undirected graph $G = (V, E)$, a *matching* is a subset of edges $M \subseteq E$ such that for each node $v \in V$, at most one edge of M is incident to v. A *maximum matching* is a matching of maximum cardinality, i.e., a matching M such that for any other matching M', we have $|M| \geq |M'|$. If all nodes are covered by the edges of a matching, such a matching is called a *perfect matching.* Clearly, one necessary condition for the existence of a perfect mapping in a graph is that the graph should have an even number of nodes. In a weighted graph, the cost of the matching is defined as the sum of costs of edges in the matching. When several matchings of the same cardinality exist in a graph, *minimum-cost* *maximum matching* is defined as a maximum matching with minimum total cost.

Given a graph and a subset of vertices, a *T-join* is defined as a subset of edges in which the degree of each vertex in the specified subset is odd and is even for all other vertices. More formally, in a graph $G = (V, E)$ and $T \subseteq V$, a *T-join* is a subset of edges E' such that, in the sub-graph $G' = (V, E')$, the degree of v is odd if and only if $v \in T$. In a connected graph $G = (V, E)$, there exists a T-join for any subset $T \subseteq V$ as long as the cardinality of T is even. A minimum-cost T-join can be computed using an algorithm for minimum-cost matchings in an auxiliary graph called the closure graph. The closure graph of G is a complete graph $H = (V, V \times V)$, where $c(u, v)$ is the cost of a shortest path between u and v. It can be shown that minimal T-joins in G correspond to minimum-cost matchings in H in which all nodes of T are matched. Refer to the book by Nemhauser and Wolsey [36] for more details on T-joins.

2.3 Flows in networks

A *flow network* $G = (V, A)$ is a directed graph in which each arc $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. Two nodes are distinguished in a flow network: a source s and a sink t. A *flow* in G is a positive-valued function f on arcs such that it satisfies the following constraints: The flow in each arc cannot exceed the arc capacity, and the total incoming flow should be equal to the total outgoing flow in all nodes except the source and the sink.

The *maximum-flow problem* is the simplest problem concerning flow networks. The objective of this problem is to find the maximum amount of flow which can be sent from the source to the sink. Ford and Fulkerson [18] presented an algorithm to solve the maximum-flow problem. When a flow network has multiple sources and multiple sinks, it can be transformed to another equivalent flow network which has a single source and a single sink, and then we can apply the standard maximum-flow algorithm.

There exists several variations of the flow problem. One of them is the *minimum-cost flow problem*, which is defined as follows: In a flow network $G = (V, A)$, each arc of A is assigned a capacity and a per-unit cost of flow. Additionally, the network has multiple sources with specified supplies, and multiple sinks with specified demands. The goal is to find a flow that exactly consumes all the supply of the source nodes and satisfies the demands of the sink nodes, and the corresponding total flow cost is a minimum. This problem can be solved by transforming it to the single source/sink maximum-flow problem.

Refer to the books by Ahuja, Magnanti, and Orlin [1], Even [16], Ford and Fulkerson [18], and Lawler [30] for elaborate discussions on network flows and related algorithms.

2.4 Properties of Eulerian graphs

A connected graph is called *Eulerian* or *unicursal* if there is a tour that traverses each edge of the graph exactly once. Here are the necessary and sufficient conditions for a connected graph to have such a tour:

- If G is undirected, every vertex must have an even degree, i.e., it must have an even number of incident edges. This condition is called the *even degree condition*. This condition implies that the number of edges crossing any cut $(S, V - S)$ is also even.

- If G is directed, the number of arcs entering a vertex must equal to the number of arcs leaving that vertex. In other words, for each vertex v, $indegree(v) = outdegree(v)$ or $surplus(v) = 0$. This condition is called the *inout degree* condition or the *symmetry* condition. This condition implies that any cut $(S, V - S)$ also satisfies the same condition.

- If G is mixed, it should satisfy the following properties [18, 37]:
  - *Even degree condition:* Every vertex v is incident to an even number of edges and arcs, i.e., $degree(v)$ is even. This condition implies that the number of edges and arcs crossing any cut $(S, V - S)$ is even.
  - *Balanced set condition:* For every nontrivial cut $S \subset V$, the absolute value of the surplus of S must be less than or equal to the number of undirected edges crossing the cut $(S, V - S)$. In other words, there should be enough number of edges across the cut to balance the net surplus created by the arcs crossing the cut.

Figure 2.1 shows an example in which the balanced set condition is satisfied across every cut. For the specified cut S, the net surplus is 2, and there are two undirected edges going across the cut to annul the imbalance. Note that the symmetry condition is a stricter version of the balanced set condition. Any mixed graph that satisfies the balanced set condition can be converted to a mixed graph that satisfies the symmetry condition using the minimum-cost flow algorithm.

2.5 NP-hard problems

Not all problems of practical importance have efficient solutions. If the solution can be computed in polynomial time, the problem is said to have an efficient solution. Problems that are not known to have efficient solutions are called intractable. One major class of such problems is the *NP-hard* class. Following are the properties of problems that are NP-complete: Validity of a given solution to the problem can be verified in polynomial time, and any other problem in NP can be transformed to it in polynomial time. Any exact algorithm that tries to solve an NP-complete problem, as of now, has an exponential running time, in terms of input size. For numerous problems with practical input size, exact algorithms cannot be run in a reasonable time even with the fastest computer. A natural question is to ask in this situation is: "Can we come up with algorithms that will output good, if not optimal, solutions?" Such algorithms employ heuristics or intuitive ideas, and are called approximation algorithms.

Researchers can come up with various heuristics for the same problem. A mechanism to compare the performance of heuristics is by comparing their *approximation ratios*. For a minimization problem, the approximation ratio is the maximum ratio between the cost of the solution output by the algorithm and the cost of optimal solution over all input instances. For a maximization problem, the approximation ratio is the minimum ratio between the cost of the solution output by the algorithm and the cost of optimal solution over all input instances. An approximation algorithm with approximation ratio α is referred to as an α-approximation algorithm. Coming up with the exact approximation ratio for a heuristic algorithm is as difficult as finding the exact solution itself, since there is no easy way to find the cost of an exact solution. So, we have to estimate the cost of an optimal solution. Based on the properties of the problem and solution, a lower bound for the cost of an optimal solution is computed and used to compute the approximation ratio. Obviously, better lower bounds will lead to lower approximation ratios. So, the lower-bound analysis is as important as good heuristics design, to come up with a good approximation ratio.

2.6 Linear programming

Most of the problems discussed in this dissertation are discrete optimization problems. These problems have a countable number of feasible solutions and the goal is to choose an optimum solution among them. The area of research concerned with the theory and techniques for these problems is called *combinatorial optimization*. One of the areas of mathematics which has extensive use in combinatorial optimization is *Linear Programming*. Linear programming (LP) can be defined as the process of optimizing (minimizing or maximizing) a linear function of variables, called the *objective function*, subject to a set of linear equalities and/or inequalities called *constraints* or *restrictions*. When the objective function and constraints are written in algebraic form, it is called a linear program. Even though linear programs can be solved using the ellipsoid method [2, 29] in polynomial time, the famous Simplex method usually exhibits good performance in practice and is extensively used to solve LP problems [28, 47].

Some optimization problems require integer values for variables in the solution. Such solutions are called as integral solutions. However, linear programming techniques do not guarantee integer solutions. A related area called *Integer Programming* studies optimization problems with integrality constraints. Additional to LP's constraints, an Integer program (IP) also has integrality constraints for the decision variables. One way of solving an IP approximately is to solve the corresponding LP relaxation, get a fractional solution, and carry out rounding to get the closest appropriate integral solution such that other constraints are not violated. This technique has been used successfully in problems such as vertex cover, max satisfiability and max-cut, but it is not guaranteed to work in all IP instances.

The following methods are widely in use to solve IP problems exactly: branch and bound, branch and cut. The main idea of these approaches is to exhaustively explore the solution space. Since the solution space can be exponentially large with respect to the problem size, a simple brute-force search approach will take too much time. Above approaches come up with an approximate solution, then proceed in branches, come up with a bound for the solution in that branch (for example, by solving the LP relaxation of the IP). If the best solution that can be arrived using that branch is worse than current solution, then that branch need not be further explored.

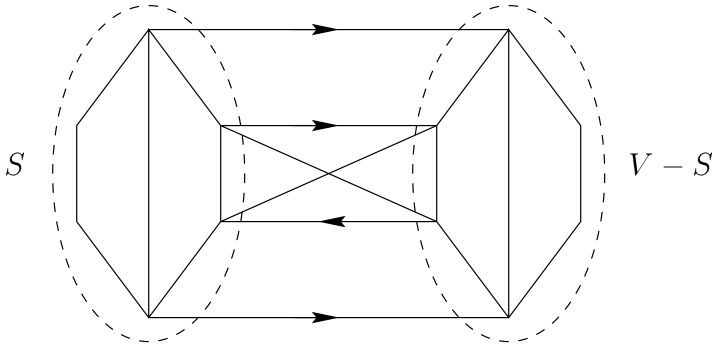


Figure 2.1: Balanced set condition

CHAPTER 3

MIXED POSTMAN PROBLEM

First, the algorithms available for the undirected postman problem, the directed postman problem, the Euler tour problem on mixed graphs, and the mixed postman problem (MPP) are surveyed. Then the core details of the NP-hardness proof for the mixed postman problem is presented. After the analysis of Frederickson's 5/3-approximation algorithm for MPP, a 3/2-approximation algorithm is presented.

3.1 Undirected postman problem

Given a weighted graph G, the Chinese postman problem (CPP) is to find a minimum cost tour covering all edges of G at least once [25]. It is the optimization version of the Euler tour problem, which asks if there is a tour that traverses every edge of a graph exactly once. We will discuss the undirected case in this section. As discussed in Chapter 2, the necessary and sufficient conditions for an undirected graph to have an Euler tour is that it is connected and that all its nodes have even degree. When there are odd-degree nodes, extra edges have to be added to make all nodes have even degree. Hence the objective of the undirected postman problem is to find extra edges of minimum total cost that can be added to the input graph such that the resulting graph has an even degree at all nodes.

Edmonds and Johnson [11] presented the first significant results for postman problems. They showed that the undirected postman problem is solvable in polynomial time. They gave two integer programs to find a minimum-cost augmentation. In the first formulation, the main constraint is that the total number of edges incident on each node should be even. They observed that this is a special case of matching problem. Undirected CPP can be reduced to the general matching problem as follows: From the input graph G, compute a complete graph G_p whose nodes are the odd-degree nodes of G, and the edge costs are the shortest path costs between corresponding nodes in G. Find a minimum-cost perfect matching in G_p. For every pair of matched nodes in this matching, add to G the edges corresponding to the shortest path between them. All nodes in G have even degree after the addition of these paths. Note that the above procedure is equivalent to adding a minimum-cost T-join of the odd-degree nodes of G. A similar algorithm was presented by Orloff [38]. In the second IP formulation of Edmonds and Johnson, the main constraint is that an odd number of additional edges are required across every cut $(S, V-S)$ in which S contains an odd number of odd-degree nodes. Since odd-degree nodes have to be matched in pairs to find a solution, at least one additional edge is required across such a cut. They called these constraints as blossom inequalities.

Figure 3.1 (a) shows an example of a graph that contains 6 odd-degree nodes. A minimum-cost T-join is added to derive the even-degree graph in Figure 3.1 (b). Note that since the algorithm has added edges (a, g) and (g, f) instead of (a, f), $C(a, g) + C(g, f)$ should be less than or equal to $C(a, f)$.

Each edge is duplicated at most once in the process of correcting the parity of nodes. If the input graph is a tree, every edge has to be duplicated in order to generate a tour. The maximum cost of an optimal T-join is at most the cost of the graph itself. When the input graph is 2-connected, the maximum cost of an optimal T-join is at most half the cost of the graph [22].

Once an even-degree graph is obtained, an Euler tour of the graph needs to be computed. Edmonds and Johnson [11] presented two algorithms to find an Euler tour in even-degree graphs. The basic idea for both algorithms is that any even-degree graph can be decomposed into a set of tours as follows: Start the tour at a random node in the graph, walk on unvisited edges orienting them along the direction of the walk, and stop when the source node is reached. The walk cannot get stuck at any intermediate node without having an unvisited edge to go out of the node because of the fact that the node has even degree. Therefore, every incoming edge at an intermediate node always has another edge with which it can be paired that enables the walk to get out of that node. The tour thus generated forms a basic Euler sub-tour. If there are any unvisited edges, restart the tour at any node that has adjacent unvisited edges and extract another sub-tour, and stitch both sub-tours at the start node of the new tour. Repeat this procedure until all edges belong to the tour.

3.2 Directed postman problem

The postman problem on directed graphs is now considered. Since any tour has to obey the direction of arcs, the directed postman problem may not have a solution even if the underlying graph is connected. As mentioned in Chapter 2, a directed graph has to be strongly connected, and symmetric, i.e., the surplus at each node should be zero, for it to have an Euler tour. Recall that the surplus of a node is the number of arcs directed towards that node minus the number of arcs directed away from that node. Note that the total surplus of the whole graph is always zero, since each arc contributes $+1$ to the surplus at one node and -1 to the surplus at another node.

Let us refer to the nodes with surplus as surplus nodes, and the nodes with negative surplus (deficit) as deficit nodes. If the input graph is not symmetric, it has one or more surplus nodes and one or more deficit nodes. The objective of the directed postman problem is to add arcs of total minimum cost such that directed paths have been added from surplus nodes to deficit nodes to make them symmetric. This explanation naturally lends itself to an algorithm based on minimum-cost flows. Orloff [38] solved the directed CPP by formulating a minimum-cost network-flow problem:

- Each node has a supply/demand equal to its surplus/deficit.
- The cost of unit-flow through the arc is its cost in G.
- Each arc is assigned unlimited capacity.

Here is the LP formulation for the directed CPP: In a given digraph $G = (V, A)$, let I be the set of nodes v_i for which the number of incoming arcs exceeds the number of outgoing arcs by s_i and let J be the set of nodes v_j for which the number of outgoing arcs exceeds the number of incoming arcs by d_j. Thus s_i can be interpreted as supply, and d_j as demand. In addition, let c_{ij} be the cost of the shortest path from v_i to v_j. Now the problem can be formulated as follows:

$$\min \sum_{v_i \in I} \sum_{v_j \in J} c_{ij} x_{ij}$$

subject to

$$
\begin{aligned}
\sum_{v_j \in J} x_{ij} &= s_i, \ \forall v_i \in I \\
\sum_{v_i \in I} x_{ij} &= d_j, \ \forall v_j \in J \\
x_{ij} &\geq 0, \ \forall v_i \in I, v_j \in J
\end{aligned}
$$

The optimal x_{ij} values represent the number of extra times each arc has to be traversed. Since the demand and supply values are integers and every arc has unlimited capacity, all flow values for the arcs are integers (without loss of generality) in a minimum-cost flow solution. The flow in each arc identifies the number of extra copies of that arc that need to be added to balance the surplus/deficit of the whole graph. Those arcs are duplicated as many times as needed and added to G to form the augmented graph G' which is symmetric at all nodes. Directed CPP also can be considered as an equivalent network flow problem of establishing a minimum-cost circulation on G subject to lower bounded arc-flow [18]. The circulation, which must be symmetric, defines the duplicate arcs to be added to G to obtain G'. Unlike the undirected CPP, the directed CPP may require that more than one extra copy of an arc be added to G. In the example in Figure 3.2, two copies of the arc labeled e have to be added to G in any postman tour.

Similar to the undirected case, we can decompose an Eulerian digraph to a collection of directed tours and stitch them together at common nodes to get a directed Euler tour. Edmonds and Johnson [11], and van Aadenne-Fhrenfest and de Brujin [49] suggested the following alternate procedure to find a tour in an Eulerian digraph:

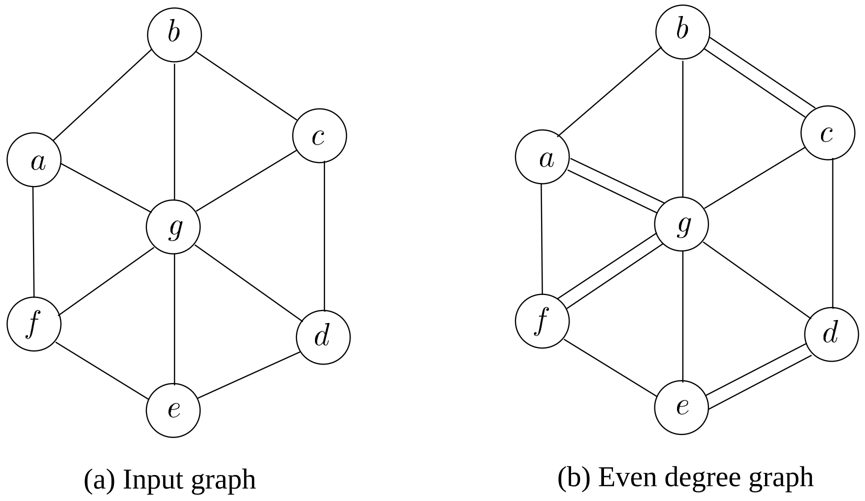


Figure 3.1: Undirected postman problem - an example

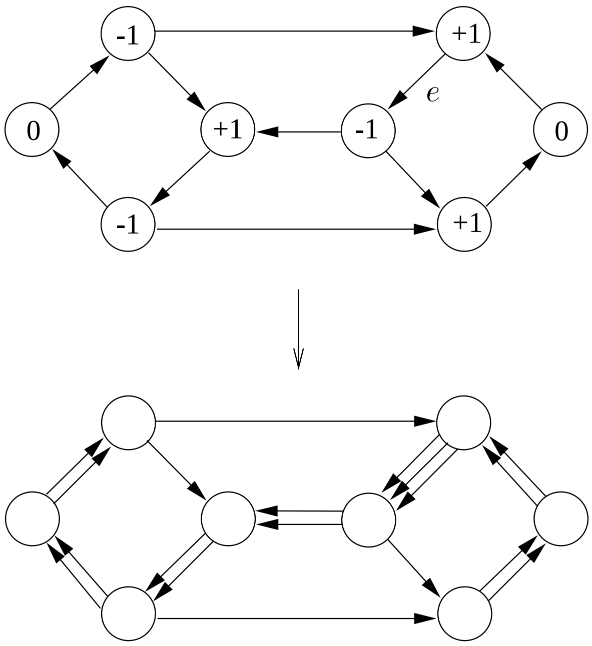


Figure 3.2: Directed postman problem - an example

- Select a root vertex v_r arbitrarily. Find a spanning arborescence rooted at v_r. Each node except v_r has exactly one outgoing arc.

- Label all arcs as follows: Order and label the arcs outgoing from v_r in an arbitrary fashion; order and label the arcs out of any other vertex consecutively in an arbitrary fashion, but keep the the arc used in the arborescence as the last arc.

- Start the Euler tour from the lowest labeled arc emanating from v_r. Whenever a vertex is entered, leave through the arc not yet traversed that has the lowest label. The procedure ends with an Euler tour when all arcs have been covered.

The label ordering mechanism used by the algorithm ensures that no arcs of the graph is missed in the process.

3.3 Mixed Euler tour problem

As specified in Chapter 2, a mixed graph should be connected and in addition satisfy the even degree condition and balanced set condition to be Eulerian. Ford and Fulkerson [18] presented the following procedure to verify whether a given mixed graph satisfies the balanced set condition. If so, the algorithm transforms it to a symmetric mixed graph:

- Find the surplus/deficit of nodes and assign them as supply/demand at nodes.

- Convert each edge to two oppositely oriented arcs, each with capacity 1. These arcs form the flow network. The original arcs of the mixed graph are not included in the flow network.

- Find a minimum-cost flow in the above network. Output contains a few arcs with a flow of 1.

- If the output satisfies all flow requirements, then the input graph satisfies the balanced set condition. If so, orient the corresponding edges accordingly in the input graph to make it symmetric. There may still be undirected cycles.

The intuitive idea behind the algorithm is to balance the mixed graph only by orienting edges. If the input graph does not satisfy the balanced set condition, there exists at least one cut in which the number of edges is less than the surplus, hence the flow cannot find a solution that satisfies all the demands.

Since the input graph also satisfies even degree condition, we have a connected, symmetric, even-degree graph at this stage. All remaining edges in the graph can be decomposed into simple cycles. Once we orient each cycle in one of its two directions arbitrarily, we will have a symmetric directed graph. Now, the algorithm for finding a directed Euler tour can be used to find an actual tour. Refer to Edmonds and Johnson [11] for another variation of this algorithm.

3.4 Mixed postman problem

Given a mixed graph $G = (V, E, A)$ with a non-negative cost function C on its edges and arcs, the objective is to find a minimum-cost tour that traverses every edge and arc at least once. Even though both undirected and directed versions of CPP are polynomially solvable, Papadimitriou [39] showed that MPP is NP-hard. MPP has been listed as problem ND25 by Garey and Johnson [23]. Since

MPP can be used to model both one-way streets and two-way streets effectively, many practical applications like mail delivery, snow removal, and trash pick-up can be modeled as instances of MPP, and hence it is important to design good approximation algorithms for this problem.

We first review the core parts of Papadimitriou's NP-hardness reduction from 3SAT to MPP before discussing the approximation algorithms. Papadimitriou designed the mixed graph shown in Figure 3.3. All edges adjacent to v have cost 2, all other edges and arcs have a cost 1 each. This graph has exactly two optimal tours. Each of the four edges adjacent to node v has opposite orientation in the two optimal solutions. In each solution, two edges are oriented towards v and the other two edges are directed away from v. Now the node is removed and the two sets of edges are used as a comparator (Figure 3.4). In an optimal tour of the structure, both edges have to be oriented inward, or both have to be oriented outward.

Several copies of the comparator structure can be combined, forcing many edges to be directed in the same way (Figure 3.5). One such gadget is constructed for each variable. One direction is associated with truth assignment value 1 for the variable, and the other direction is associated with truth assignment value 0 for the variable. Another structure (Figure 3.6) is designed for each clause. The node w is common for all clause-structures, and three edges emanate from the structure, and they are connected to the gadgets of variables that occur in that clause. If the symbol is x_i in the clause, the corresponding edge is to connected to the left side of the gadget, and if the symbol is $\neg x_i$, the corresponding edge is connected to the right side of the gadget. This structure has an optimal solution of 0 cost when the clause is satisfied, i.e., it has at least one symbol with value 1.

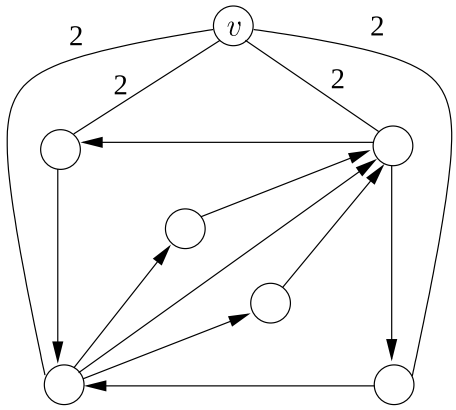


Figure 3.3: Mixed graph with exactly 2 optimal tours

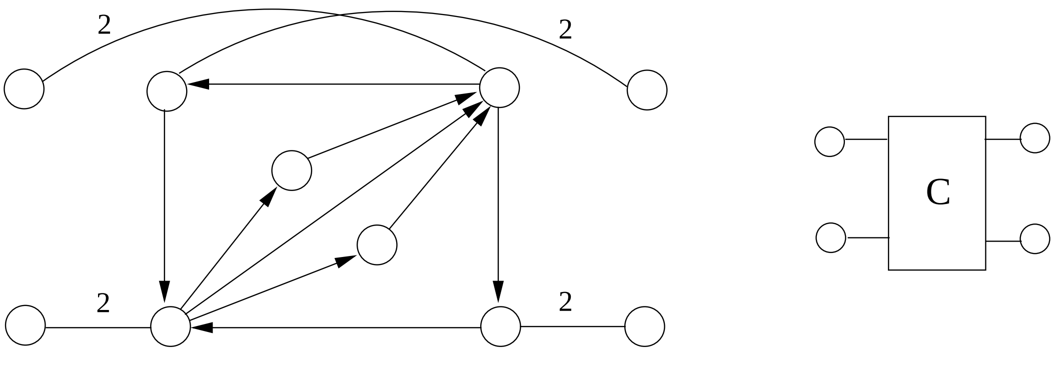


Figure 3.4: Comparator structure from mixed graph

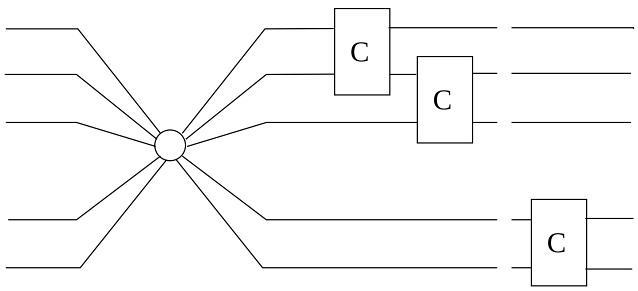


Figure 3.5: Gadget for each variable of 3SAT

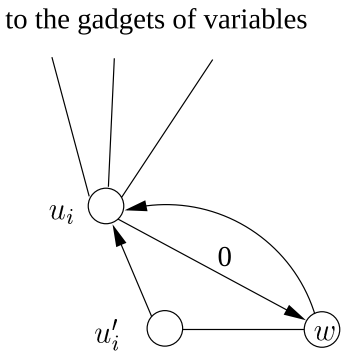


Figure 3.6: Gadget for each clause of 3SAT

When a clause is not satisfied, all edges that correspond to entries of the clause will be outgoing, and it costs more to balance u_i.

Hence a satisfying truth assignment will result in an optimal solution for the mixed graph. Similarly, if the optimal tour costs exactly twice the number of comparators, it can be mapped to a valid truth assignment. For any suboptimal MPP solution, at least one clause is not satisfied, or at least one variable takes both values. It can be shown that MPP is NP-complete even for the following special cases:

- The underlying graph is planar and the degree of each node is limited to 3 and the costs of all edges is equal to 1 [39].
- The number of traversals for each arc/edge is restricted to at most two, i.e., each arc/edge is duplicated at most once [48].

Numerous articles have appeared in the literature over the past three decades about the mixed postman problem. Edmonds and Johnson [11] presented necessary and sufficient conditions for the existence of a postman tour in a mixed graph:

- Graph should be connected.
- There is no proper subset S of nodes such that the cut $(S, V-S)$ contains only arcs from S to $V-S$ (i.e., no arcs are going from $V-S$ to S, and no edges crossing the cut).

If such a cut exists, it is impossible to balance it. Hence a postman tour does not exist in such graphs. Edmonds and Johnson also presented an optimal algorithm for MPP on even-degree mixed graphs. They gave a minimum-cost flow formulation to augment the graph to make it symmetric, while preserving the even-degree condition. An Euler tour can be found in symmetric, even-degree graphs by the algorithm presented in Section 3.3.

In the case of general mixed graphs, Edmonds and Johnson suggested to solve undirected CPP first to convert it to an even-degree graph, and then run their algorithm for even-degree graphs. Frederickson [20] showed that such an algorithm finds a tour whose length is at most twice the length of an optimal tour (i.e., approximation ratio of 2). He also presented a mixed strategy algorithm, which used the solutions output by two different heuristics, and then selected the shorter of the two tours. He proved that the approximation ratio of the mixed strategy algorithm is 5/3. Brucker [4] and Eiselt *et al* [12] have written comprehensive surveys on algorithms for postman problems.

Ralphs [44] showed that a linear relaxation of MPP has optimal solutions that are half-integral. One could use this to derive a 2-approximation algorithm for the problem. Nobert and Picard [37], and Grötschel and Win [24] have used integer programming to solve MPP. In the formulation used by Nobert and Picard, there is only one variable y_{ij} associated with every edge of E. So, the IP solution does not therefore specify edge directions. Constraints are imposed such that the augmented graph satisfies the necessary and sufficient conditions for being an Eulerian graph (Section 2.4). Since the specification of the balanced condition requirement for all cuts results in an exponential number of constraints, Nobert and Picard describe a procedure to identify the most unbalanced sets and use them to solve the LP. In each iteration, constraints that are violated are added. When no LP constraint is violated, a number of Gomory cuts are added if needed, to regain integrality of the solution. If the solution is integral and satisfies all constraints, an optimal solution has been found. Otherwise, a branching process is initiated. The algorithm was applied to a large number of randomly generated MPP's, and they reported that roughly 3 out of 4 problems were solved without the need for branching. Their implementation has been used for scheduling snow removal in Montreal.

3.5 Our results

Even though numerous articles have appeared in the literature on MPP after Frederickson's paper in 1979, his result has been the best approximation algorithm for MPP in terms of proven worst-case ratio until now. In this dissertation, we present an improved approximation algorithm for MPP with an approximation ratio of 3/2. We study the properties of feasible solutions to MPP, and derive a new lower bound on the cost of an optimal solution. Our algorithm uses a subtle modification of Frederickson's algorithm, and the improved performance ratio is derived from the new lower bound.

3.6 Frederickson's Mixed algorithm

We will summarize Frederickson's algorithm before presenting the new approximation algorithm. Frederickson defined the following algorithms as part of his solution to MPP:

- EvenDegree: Augment a mixed graph G by duplicating edges and arcs such that the resulting graph has even degree. A minimum-cost solution is obtained by disregarding the directions of the arcs (i.e., by taking the underlying undirected graph) and solving CPP by adding a minimum-weight

T-join of all odd-degree nodes.

- INOUTDEGREE: Augment a mixed graph G by duplicating edges and arcs, and orienting edges such that in the resulting graph, for each vertex v, $surplus(v) = 0$. We will refer to this as the INOUT problem. A minimum-cost solution G_{IO} is obtained by formulating a flow problem and solving it optimally. The flow problem is formulated such that the original undirected edges can be oriented free of cost. The *augmentation cost*, $C_{IO}(G)$ is defined as the cost of additional edges and arcs that are added to G to get G_{IO} by INOUTDEGREE.

  Here is the brief description of flow problem used by Frederickson [20] for the INOUT problem: Let $G = (V, E, A)$ be the input graph, where E is a set of undirected edges and A is a set of arcs. A flow network $N = (V, E')$ is created as follows. $E' = A \cup E_1 \cup E_2$, where E_1 and E_2 are defined as follows. For each edge $\{v, w\} \in E$, we add the arcs (v, w) and (w, v) to E_1 and also to E_2.

$$\min z = \sum_{s \in A} c_s x_s + \sum_{s \in E_1} c_s x_s$$

subject to

$$x_s \leq 1, \text{ for all } s \in E_2 \tag{3.1}$$

$$\sum_{w \in V} x_{vw} - \sum_{u \in V} x_{uv} = b_v, \text{ for all } v \in V, \tag{3.2}$$

$$x_s \geq 0, \text{ for all s} \in A \cup E_1 \cup E_2. \tag{3.3}$$

The arcs of E_2 are used to represent the fact that the edges of E can be oriented for free by the flow problem. Therefore E_2 does not figure in the objective function, but the capacity of its edges is limited to 1 (constraint (3.1)). The arcs of E_1 represent the edges of E that are duplicated and oriented by the flow problem. Hence the variables corresponding to E_1 appear in the objective function. Each node v has an imbalance of b_v, which is defined as the number of arcs in A that are directed towards v minus the number of arcs in A that are directed away from v. Constraint (3.2) ensures that the solution to the flow problem corrects this imbalance at each vertex. It is not necessary to specify integrality conditions on the variables, since basic optimal solutions to flow problems are always integral. It can be shown that an integer optimal solution to the above flow problem can be found using standard algorithms for the minimum-cost flow problem.

- EVENPARITY: applied to the output of INOUTDEGREE on an even-degree graph; restores even degree to all nodes without increasing the cost, while retaining the property that *indegree* = *outdegree* at all nodes. Edmonds and Johnson [11] indicated that INOUTDEGREE can be applied to an even-degree graph in such a way that the resulting graph has even degree and hence Eulerian. Frederickson [20] showed a simple linear-time algorithm to perform the task. The basis of this algorithm is that a suitably defined subgraph of undirected edges and duplicated edges/arcs forms a collection of Eulerian graphs.

- LARGECYCLES: Similar to EVENDEGREE except that only edges are allowed to be duplicated, and arcs are not considered.

Frederickson [20] presented an approximation algorithm for MPP called MIXED algorithm (see Figure 3.7). The algorithm comprises of two heuristics called MIXED1 and MIXED2. Heuristic MIXED1 first runs EVENDEGREE to make the degree of all nodes even. Then it runs INOUTDEGREE to make *indegree* = *outdegree* for all vertices. Finally, EVENPARITY restores even degree to all the nodes without increasing the cost, and the graph becomes Eulerian. Heuristic MIXED2 first calls INOUTDEGREE, and then makes the graph Eulerian by calling LARGECYCLES. Since LARGECYCLES does not duplicate arcs of the graph, no further steps are needed. The MIXED algorithm outputs the best solution of the two heuristics, and Frederickson showed its performance ratio is at most 5/3. Brucker [4] presented another solution to the INOUTDEGREE problem using minimum-cost flow, and his solution is similar to that of Edmonds and Johnson [11]. Unlike INOUTDEGREE algorithm, given an even-degree input graph, his algorithm preserves the even degree in the output, so the final step EVENPARITY is not required. A version of Brucker's algorithm is presented in Chapter 4.

3.7 Structure of INOUT problem

In this section, we will identify a few critical properties of the INOUT problem, which we use to improve the MIXED algorithm. Let G be a mixed graph, and let G_b and G_r (blue graph and red graph respectively) be two augmentations of G for the INOUT problem, i.e., G_b and G_r are two INOUT solutions.

Let $G_{br} = G_b \oplus G_r$ denote the *symmetric difference* of G_b and G_r, the multigraph containing edges and arcs in one graph, but not the other. Since we are dealing with multigraphs, an edge/arc (u, v), occurs in G_{br} as many times as the difference between the number of copies of (u, v) in G_b and G_r. A set of arcs C in G_{br} is an *alternating cycle* if the underlying edges of C form a simple cycle, and in any walk around the cycle, the blue arcs of C are directed one way, and the red arcs of C are directed in the opposite direction (see Figure 3.8).

The following lemmas show how alternating cycles of G_{br} can be used to switch between different INOUT solutions.

Lemma 3.1 *The arcs of G_{br} can be decomposed into alternating cycles.*

Proof. This proof is based on the properties of Eulerian directed graphs. Since they are INOUT solutions, each vertex of G_b and G_r satisfies $in\_degree = out\_degree$. However, this rule does not apply to G_{br}, since all common arcs and edges have been removed. Since only common arcs are removed, it can be verified that for each vertex in G_{br}, $surplus_b = surplus_r$. In other words, at each vertex of G_{br}, the net surplus or deficit created by blue arcs is equal to the net surplus or deficit created by red arcs. So at each vertex, an incoming blue arc can be associated with either an outgoing blue arc, or an incoming red arc. A similar statement is valid for red arcs also.

Consider a directed walk starting at an arbitrary vertex in G_{br}, in which blue edges are traversed in the forward direction and red edges are traversed in the reverse direction. Whenever any vertex is revisited during the walk, an alternating cycle is removed and output. When the start vertex is re-visited and there are no more arcs incident to it, restart the walk from another vertex. Continue the process until all arcs of G_{br} have been decomposed into alternating cycles. An alternative proof can be given by showing that reversing all red arcs makes G_{br} satisfy the symmetry condition at each node and hence the arcs can be decomposed into cycles. ∎

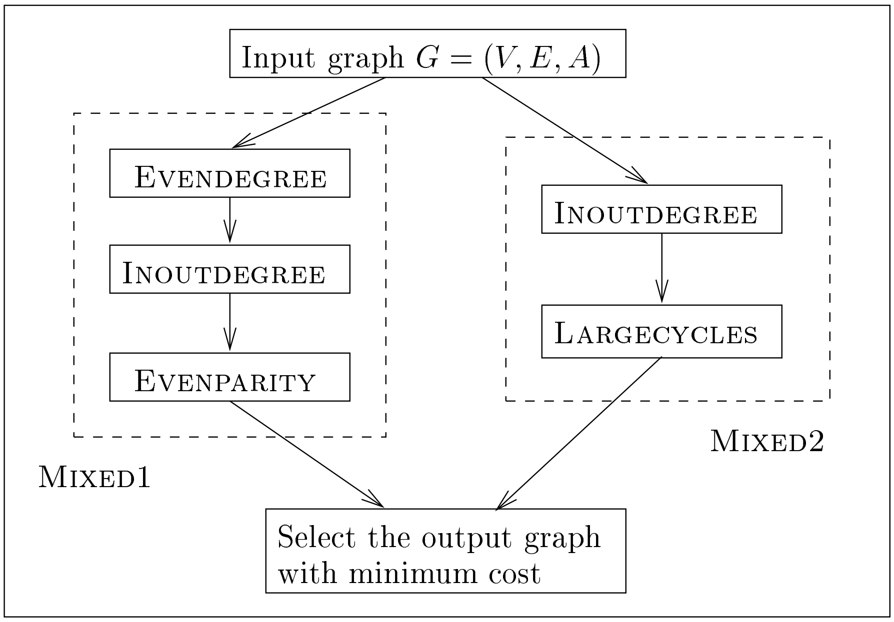


Figure 3.7: MIXED algorithm

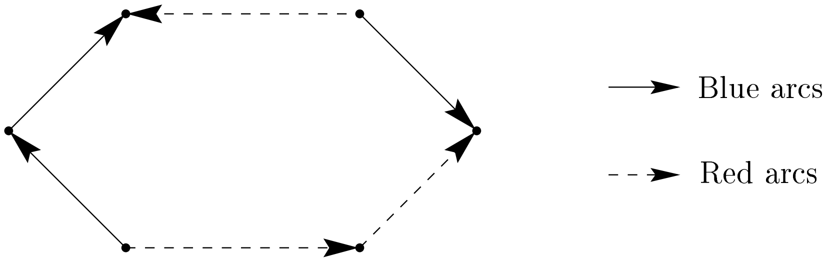


Figure 3.8: An alternating cycle

Each alternating cycle AC in G_{br} is composed of zero or more blue paths and red paths. In each cycle, all blue paths go in one direction and all red paths go in the opposite direction. Let us call blue arcs of AC as B and red arcs as R.

Define the operation of removing a set of arcs B from an INOUT solution G_{IO}, denoted by $G_{IO} - B$, as follows. If an arc of B is an additional arc or edge added during the augmentation, then it can be removed. On the other hand, if the arc was an undirected edge in G that was oriented during the augmentation, then we remove the orientation but leave the edge in the graph. The cost of such an operation is the total cost of the arcs/edges that are removed. Similarly, we can define the addition operation $G_{IO} + B$, where a set of arcs B are added to G_{IO}.

Lemma 3.2 *Consider two* INOUT *solutions G_b and G_r, and their symmetric difference G_{br}. Let AC be an alternating cycle in G_{br}, with blue component B and red component R. Then, $G_b - B + R$ and $G_r - R + B$ are also* INOUT *solutions.*

Proof. Clearly, adding or deleting edges of AC does not affect the deficit/surplus of the nodes that are not in AC. For all nodes in AC, the net deficit/surplus created by blue paths of AC is same as the net deficit/surplus created by red paths of AC. So, the net deficit/surplus created by removal of B from G_b is compensated by the addition of R. Therefore, $G_b - B + R$ is an INOUT solution, as each node of $G_b - B + R$ has net deficit/surplus of zero. By symmetry, $G_r - R + B$ is also an INOUT solution. ∎

Lemma 3.3 *Let $d^-(R)$ be the cost saved by removing R from G_r and let $d^+(R)$ be the cost of adding R to G_b. Similarly, let $d^-(B)$ be the cost saved by removing*

B from G_b and let $d^+(B)$ be the cost of adding B to G_r. Then $d^+(R) \leq d^-(R)$ and $d^+(B) \leq d^-(B)$.

Proof. We prove that the cost of adding R to G_b does not exceed the cost saved by removing R from G_r. Arcs of R in G_r can be either additional arc, oriented edge, or additional oriented edge. The contribution of an additional arc or an additional oriented edge is equal to its cost and therefore is the same in $d^+(R)$ and $d^-(R)$. If an oriented edge is in R, then either it is not currently oriented in G_b or it is oriented in the opposite direction in G_b. To add this oriented edge to G_b, we can either orient the undirected edge, or remove its opposite orientation; either way there is no additional cost. Therefore R can be added to G_b without incurring additional cost. Hence, $d^+(R) \leq d^-(R)$. By symmetry, $d^+(B) \leq d^-(B)$. ∎

3.8 Improved lower bound for mixed postman problem

Consider a mixed graph $G = (V, E, A)$, with edges E and arcs A. Let C^* be the weight of an optimal postman tour of G. Suppose, given G as input, InOutDegree outputs $G_{IO} = (V, U, M)$. $U \subseteq E$ are the edges of G that were not oriented by InOutDegree. $M \supseteq A$ are arcs that satisfy the symmetry condition at each vertex. Let C_M and C_U be the total weight of M and U respectively. Note that $C_M + C_U = C_{IO}(G) + C(G)$. Consider the graph induced by M. It contains one or more strongly connected components when M is not empty. Since all nodes in G_{IO} have surplus zero, each arc of M is in one of the strongly connected components. We will refer to such components as *directed components*. Let us define an equivalence relation Q on the nodes of G as follows: For any two nodes u and v, uQv if and only if u and v are in the same directed component in G_{IO}. It can be verified that Q is an equivalence relation. Suppose we shrink G by replacing

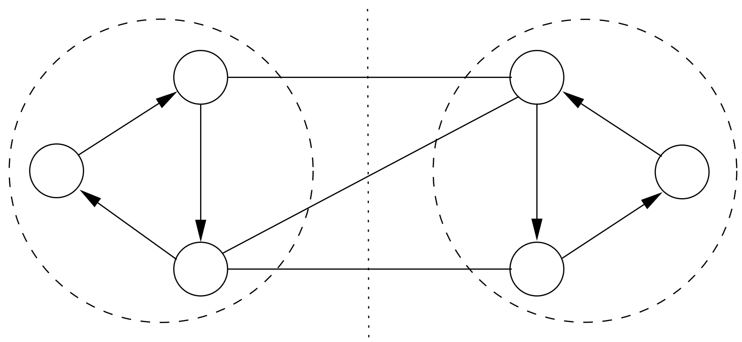


Figure 3.9: Supernodes with an odd number of odd-degree nodes

the nodes of each equivalence class of Q by a single node. Let us denote the output multigraph as UG. Note that UG can also be derived from G_{IO} simply by shrinking each directed component to a single node. Since all arcs are part of directed components, there are no arcs in UG. We will refer to the nodes of UG as *supernodes* for clarity. Any node not adjacent to arcs in M is itself a directed component, and hence a supernode of UG.

If all supernodes of UG are of even degree, we can find a T-join of all odd-degree nodes of G by duplicating only the arcs and edges of M, and in this case it is possible to achieve an approximation ratio of 3/2 using Frederickson's analysis. However, as shown in Figure 3.9, if there are supernodes of UG that have odd degree, i.e., if there are equivalence classes of Q with an odd number of odd-degree nodes in them, then a T-join of odd-degree nodes is forced to use a subset of U. We need to improve the lower bound to account for these edges in order to achieve a 3/2 approximation ratio for this case.

Consider a T-join T of the odd-degree nodes of G. The degree of each vertex in the union of T and G is even. Hence every cut in it is crossed by an

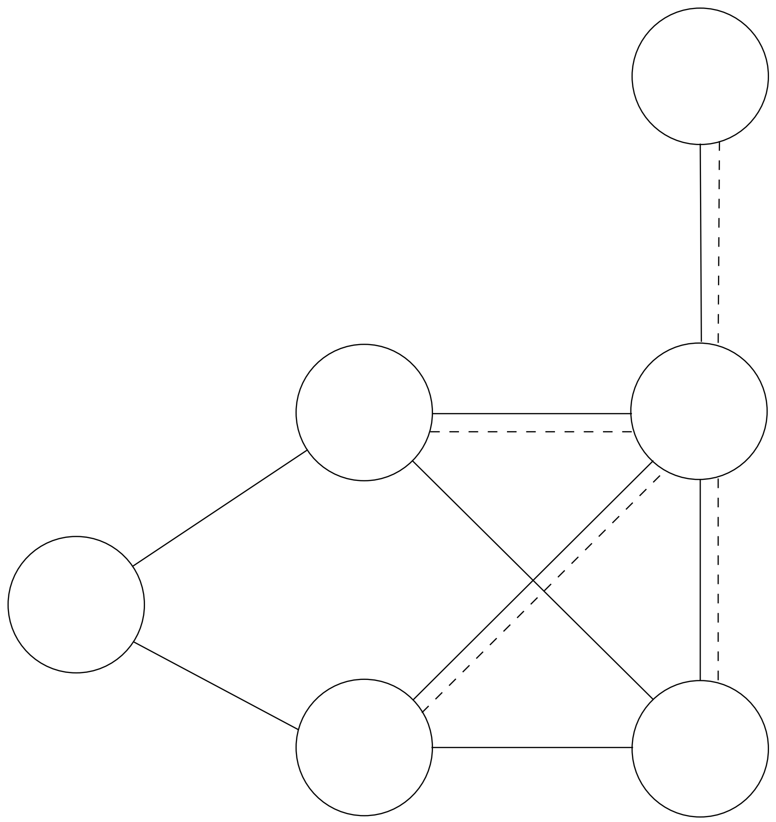


Figure 3.10: T-join edges between the odd degree supernodes

even number of edges. Odd-degree supernodes in UG correspond to cuts in G that are crossed by an odd number of edges. Since the addition of T makes the number of edges crossing each of these cuts to be even, it can be verified that the set of edges of T that connect different supernodes forms a T-join of the odd-degree supernodes of UG. Also, if a T-join of the odd-degree supernodes of UG is added to G_{IO}, the number of odd-degree vertices in each of its directed components becomes even, and therefore the resulting graph can be made to have even-degree by duplicating edges of M alone. Figure 3.10 shows an example in which the T-join edges are represented by dashed lines. Addition of these edges make every supernode have even degree. We show that Frederickson's lower bound on the optimal cost of MPP can be improved to include the cost of a minimum-weight T-join of the odd-degree supernodes of UG.

If we run EVENDEGREE on UG, it adds a minimum-cost T-join of odd-degree nodes (i.e., a subset of U) to make the degree of each supernode in UG even. Let the cost of this T-join be C_X. Frederickson [20] used a lower bound of $C_M + C_U$ on C^*. We show the following improved lower bound on C^*, which allows us to improve the approximation ratio.

Lemma 3.4 $\quad C_M + C_U + C_X \leq C^*$.

In order to prove the above lemma, we first show that when we find an optimal INOUT solution, if some undirected edges are not oriented (edges in the set U) by INOUTDEGREE, then adding additional copies of edges in U to G does not decrease $C_{IO}(G)$, the augmentation cost of INOUTDEGREE.

Lemma 3.5 *Let $G = (V, E, A)$ be a mixed graph. Let $G_{IO} = (V, U, M)$ be an optimal* INOUT *solution computed by* INOUTDEGREE *algorithm. Let $U = \{u_1, ..., u_k\}$*

be the undirected edges of G that were not oriented by InOutDegree*, even though the algorithm could orient these edges without incurring additional cost. Adding additional copies of edges in U to G does not decrease the augmentation cost of an optimal* InOut *solution.*

Proof. We give a proof by contradiction. Let H be the graph G to which n_i copies of u_i $(i = 1, \ldots, k)$ are added such that the augmentation cost of InOutDegree is less for H than G, i.e., $C_{IO}(H) < C_{IO}(G)$. Let Y be $H - G$. In addition, let H be a minimal supergraph of G with this property, and if H contained fewer copies of the edges of U, then $C_{IO}(H) = C_{IO}(G)$. Clearly, some edges of U must be oriented in H_{IO}, otherwise an InOut solution cheaper than G_{IO} can be derived from H_{IO} for G. Let G_r be H_{IO}. Construct another InOut solution for H, by adding Y to G_{IO}. Let it be G_b. We now consider $G_{br} = G_b \oplus G_r$ and properties of alternating cycles in G_{br}. We will prove that $d^-(R) < d^-(B)$ for every cycle AC that uses at least one edge of U. Recall that $d^-(R)$ is the cost of removing R from the InOut solution G_r, and $d^-(B)$ is the cost of removing B from the InOut solution G_b.

Consider any alternating cycle AC that contains an edge of U. Let B and R be its blue and red arcs respectively. By Lemma 3.3, $d^+(B) \leq d^-(B)$. If $d^-(R) \geq d^-(B)$ then,

$$C(G_r - R + B) = C(G_r) - d^-(R) + d^+(B) \leq C(G_r).$$

Therefore, $G_r - R + B$ is an optimal InOut solution of H, and it uses fewer copies of edges of U than H. This contradicts the minimality assumption of H. Therefore $d^-(R) < d^-(B)$ for each cycle AC that involves at least one edge from U. Choose any such cycle AC. We already know that $d^+(R) \leq d^-(R)$ and $d^-(R) < d^-(B)$. Combining these, we get,

$$C(G_b - B + R) = C(G_b) - d^-(B) + d^+(R) < C(G_b).$$

Note that $G_b - B + R$ uses at most one copy of each edge from U since each alternating cycle is a simple cycle. So, it is possible to remove Y from $G_b - B + R$ to derive an InOut solution for G.

$$C(G_b - B + R) < C(G_b) \Rightarrow C(G_{IO} - B + R) < C(G_{IO})$$

In other words, we have derived a new InOut solution $G_{IO} - B + R$ for G, and it costs less than G_{IO}. This contradicts the assumption that G_{IO} is an optimal InOut solution of G. Therefore, adding additional copies of edges from U does not decrease the augmentation cost of an InOut solution. ∎

Remark: It can be verified that Lemma 3.5 is true even when B is empty in some alternating cycles of G_{br}.

Proof of Lemma 3.4. Let G^* be an optimal solution, whose cost is C^*. Consider the supernodes of UG, which correspond to the directed components of M. G^* needs to have additional T-join edges (subset of U) between these components to satisfy the even-degree condition for each component. This T-join costs at least C_X. By Lemma 3.5, we know that additional edges of U do not decrease the cost of optimal InOut solution. Therefore the augmentation cost of the InOut problem is still $C_{IO}(G)$, after adding T-join edges to input graph G. Hence the total cost of G^* is at least $C_X + C_{IO}(G) + C(G)$. Substituting $C_M + C_U$ for $C_{IO}(G) + C(G)$, we get, $C_M + C_U + C_X \leq C^*$. ∎

3.9 Modified Mixed algorithm

Figure 3.11 describes the modified Mixed algorithm. First, we run algorithm InOutDegree on input graph G and obtain $G_{IO} = (V, U, M)$. Before running EvenDegree of Mixed1 algorithm, reset the weights of all arcs and edges used by M to 0, forcing EvenDegree to duplicate edges/arcs of M whenever possible, as opposed to duplicating edges of U. In other words, use the following cost function when running EvenDegree algorithm:

$$c'(e) = \begin{cases} 0 & \text{if } e \in M \\ c(e) & \text{otherwise.} \end{cases}$$

Use the actual weights for the rest of the Mixed1 algorithm. There are no changes made in Mixed2 algorithm.

Remark: The cost of arcs and edges of M need not be set to zero. In practice, the cost of each edge in U could be scaled up by a big constant. This ensures that the total cost of edges from U in the minimum-cost T-join is minimized.

3.9.1 Analysis of Modified Mixed algorithm

Lemma 3.6 *Consider the minimum cost T-join of odd degree supernodes of UG. After adding this T-join to G, compute a T-join of odd degree nodes within each equivalence class. These steps together are equivalent to computing the minimum cost T-join of odd-degree nodes of G with the new cost function c'.*

Proof. The minimum cost T-join of odd degree supernodes of UG contains a few edges from U and costs C_X. Once we add these edges to G, each equivalence class of G has an even number of odd-degree nodes. Since the nodes in each equivalence class are connected, there exists a T-join of the odd-degree nodes

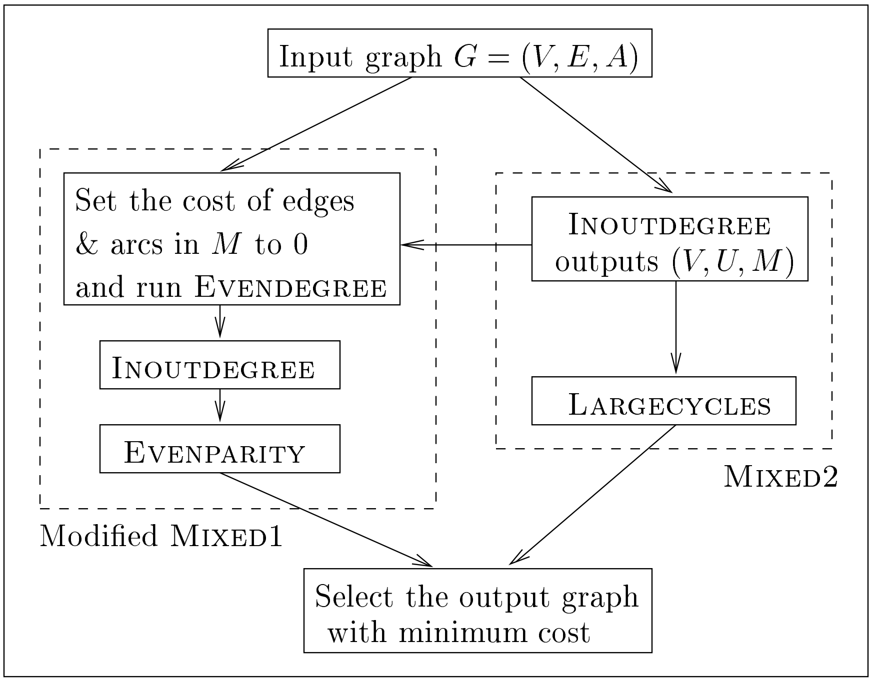


Figure 3.11: Modified MIXED algorithm

within each equivalence class using only the edges and arcs of M. These two steps together make all nodes of G even degree. Now consider the execution of EVENDEGREE on G with the new cost function c': Since EVENDEGREE is an optimal algorithm, it computes a minimum cost T-join of odd-degree nodes of G. As observed earlier, it induces a T-join of the odd-degree supernodes of UG and therefore costs at least C_X. ∎

Lemma 3.7 *Let C_M be the cost of arcs in M and let C^* be the cost of an optimal postman tour of G. The cost of the tour generated by Modified* MIXED1 *algorithm is at most $C^* + C_M$.*

Proof. Consider the components induced by the arcs of M. In the original graph G, the arcs of M correspond to arcs and oriented edges, possibly duplicated. By setting the cost of edges and arcs of M to 0 in G, we force EVENDEGREE not to duplicate the edges of U as much as possible. By Lemmas 3.4 and 3.6, its output graph will contain a few duplicated edges of U (costing C_X), and a few duplicated edges and arcs of M. Let H be the graph at this stage. Note that EVENDEGREE duplicates each arc of M at most once to form H. We follow Frederickson's analysis [20] for the rest of the proof: Let M_1 contain two copies of each arc in M. Clearly, M and M_1 both satisfy INOUT property. Hence, the union of U, X, and M_1 forms a INOUT solution containing H, whose cost is $C_U + C_X + 2 * C_M$. Since INOUTDEGREE is an optimal algorithm for INOUT problem, it is guaranteed to find an INOUT solution of cost at most $C_U + C_X + 2 * C_M$. This is at most $C^* + C_M$ by Lemma 3.4. Finally, EVENPARITY does not change the cost of the solution. ∎

Lemma 3.8 (Frederickson [20]) *Algorithm* MIXED2 *finds a tour whose cost is at most* $2C^* - C_M$.

Proof. Recall that algorithm MIXED2 runs algorithm INOUTDEGREE followed by algorithm LARGECYCLES. Consider the output of algorithm INOUTDEGREE $G_{IO} = (V, U, M)$. According to Lemma 3.4, $C_M + C_U + C_X \leq C^*$. So, $C_U \leq C^* - C_M$. Since algorithm LARGECYCLES may duplicate U in the worst case, the maximum cost of MIXED2 solution is $C_M + 2(C^* - C_M) = 2C^* - C_M$. ∎

Theorem 3.1 *Algorithm Modified* MIXED *produces a tour whose cost is at most* $1.5C^*$.

Proof. By Lemma 3.7, Modified MIXED1 outputs a solution whose cost is at most $C_M + C^*$, which is at most 1.5 C^*, if $C_M \leq C^*/2$. On the other hand, if $C_M > C^*/2$, then by Lemma 3.8, MIXED2 outputs a solution whose cost is at most $2C^* - C_M$, which is at most 1.5 C^*. ∎

3.10 Worst-case examples

The worst-case example presented by Frederickson [20] for algorithm MIXED1 contains a minor error. Figure 3.12 shows the corrected example. The input graph is shown in (a). The output of algorithm EVENDEGREE is shown in (b). The output of algorithm INOUTDEGREE is shown in (c). The output of algorithm EVENPARITY, hence the output of algorithm MIXED1 (and of Modified MIXED1) is shown in (d). Finally, the optimal solution is shown in (e). So, the MIXED1 solution costs $4 + 8\epsilon$ and the optimal solution costs $2 + 10\epsilon$. As ϵ tends to zero, the performance ratio of MIXED1 algorithm reaches 2.

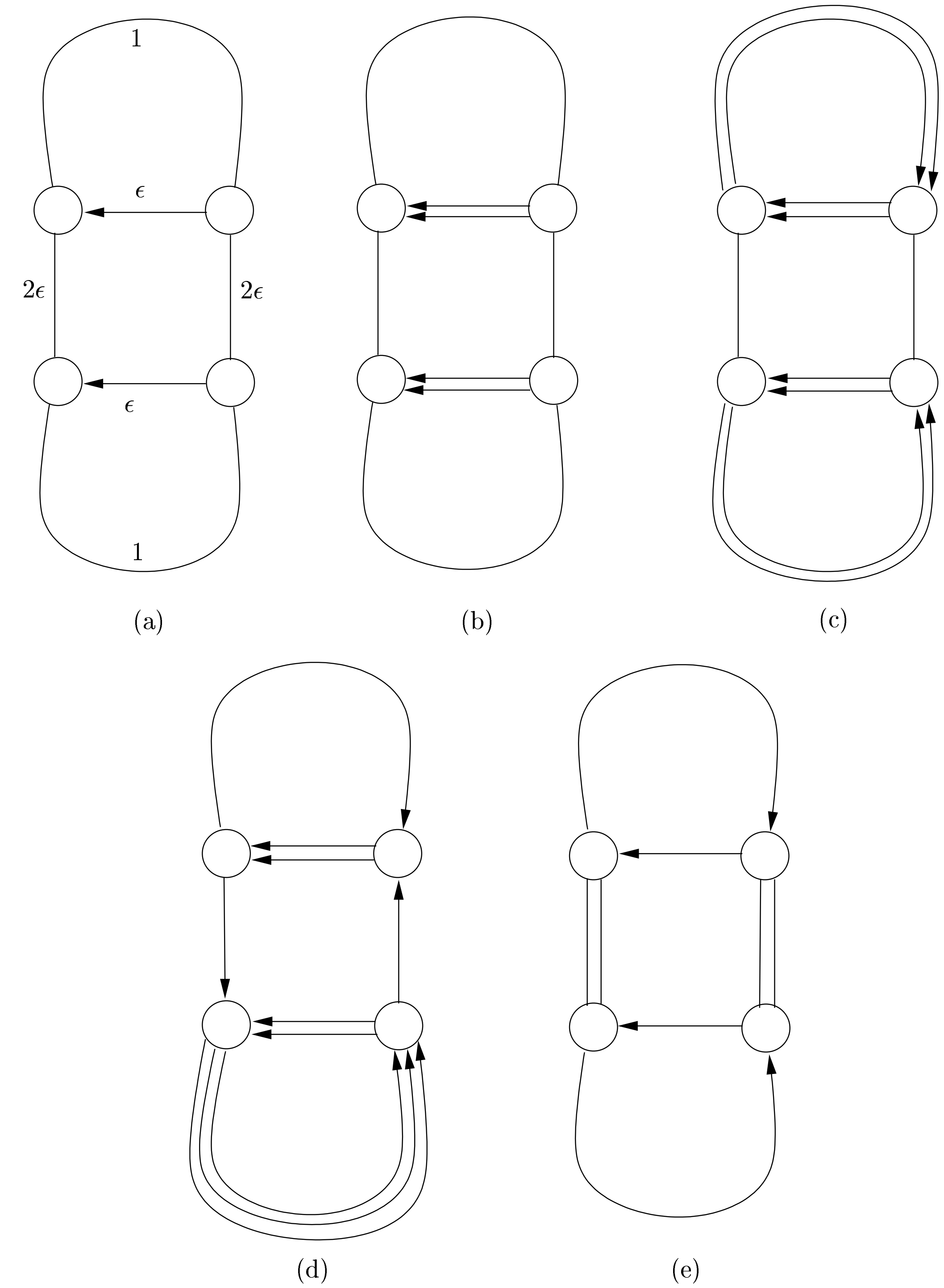


Figure 3.12: A worst-case example for algorithm MIXED1

Figure 3.13 shows the worst-case example for algorithm Mixed2 given by Frederickson. The input graph is shown in (a). Algorithm Inoutdegree does not make any changes since the graph is already symmetric. Algorithm Largecycles duplicates the edges and the output graph is shown in (b). The optimal solution is shown in (c). So, the output to Mixed2 costs $4 + 2\epsilon$ and the optimal solution costs $2 + 3\epsilon$. As ϵ tends to zero, the performance ratio of Mixed2 algorithm reaches 2.

We combine these examples to form a worst-case example for algorithm Modified Mixed and it is shown in Figure 3.14. When ϵ tends to 0, Modified Mixed algorithm outputs a solution with cost 6, the optimal solution costs 4, and their ratio is 1.5. This example proves that our analysis of Modified Mixed algorithm is tight.

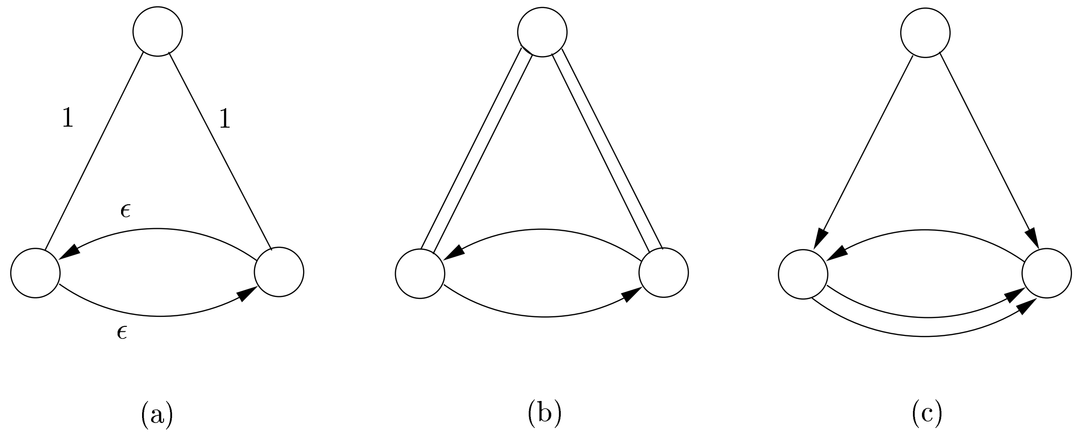


Figure 3.13: A worst-case example for algorithm MIXED2

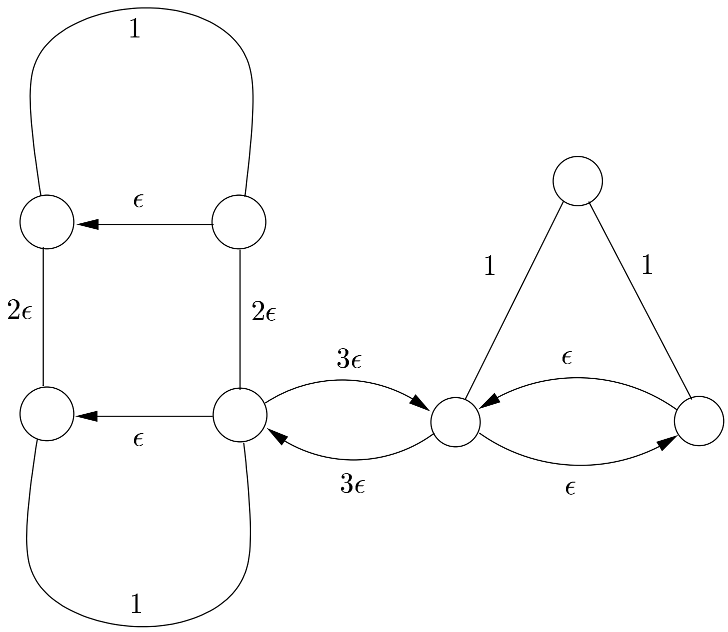


Figure 3.14: A worst-case example for Modified MIXED

CHAPTER 4

WINDY POSTMAN PROBLEM

In all variations of the postman problems considered up to now, the cost of traversing an edge is the same in both directions. This may change when one direction is uphill and the other downhill, or when one direction is with the wind and the other against, or when the fares are different depending on the direction. The *Windy postman problem* (WPP), a generalization of CPP, captures such situations. In this formulation, the cost of traversing an edge in one direction can be arbitrarily different from the cost of traversing the edge in the opposite direction. Note that the underlying graph is undirected, so the edges can be traversed in either direction, but with different costs. This problem is also NP-hard [4, 26, 39]. Undirected postman, directed postman, and mixed postman problems are all special cases of WPP.

4.1 Problem statement

Given an undirected graph $G = (V, E)$, where V is a set of vertices, and E is a multi-set of edges, and an asymmetric, positive cost function $c = (c_{ij}, c_{ji})$ for each edge (i, j), the problem is to find a minimum-cost tour traversing each edge at least once. Note that, for each edge (i, j), c_{ij} can be arbitrarily different from

c_{ji}. We assume that the graph is connected, i.e., there exists a path from any vertex u to any vertex v, since the problem is clearly infeasible for graphs that are not connected. The output is a tour which may traverse each edge several times. Therefore, except for traversing each edge once, the traversal could always use a shortest path between any two nodes, similar to the solutions to the other postman problems.

4.2 Preliminaries

Intermediate solutions of WPP may include undirected edges, duplicated undirected edges, oriented edges, and duplicated oriented edges. When there is no ambiguity, we will refer to undirected edges as edges, and oriented edges as arcs. Let the average cost of an edge (i, j) be the average of its two directional costs, $(c_{ij} + c_{ji})/2$. Let the orientation cost of an arc (i, j) be its directional cost minus the average cost of the corresponding edge, $(c_{ij} - c_{ji})/2$.

Let the cost of an intermediate solution be the sum of the directional costs of its arcs, and the average cost of its edges. This provides a measure for evaluating solutions at various stages of the algorithm. Based on this definition, the cost of the input graph is the sum of the average cost of its edges, since it has no arcs. The cost of the final solution graph is the sum of the directional costs of its arcs, since it has no edges. Observe that unlike CPP and MPP, the cost of the input graph is not a lower bound on the cost of an optimal solution to the WPP.

4.3 Previous work

Minieka [35] introduced the WPP in 1979. Guan [26] proved that WPP is NP-hard by presenting a following reduction from the mixed postman problem (MPP) to WPP. Given a mixed graph G, keep each edge in the output graph with the same cost in both directions. For each arc (x, y) in G, introduce an undirected edge (x, y) with directional cost c_{xy} for (x, y) and a very large cost M for (y, x). Figure 4.1 shows an example mapping. We have specified the directional costs near the tail end of corresponding arcs in WPP graphs. Now, solving WPP in the new graph solves the mixed postman problem in G. It is straight-forward to see that the mixed postman problem on G has a feasible solution if and only if the corresponding WPP has a solution that costs less than M.

Guan [26] presented an approximation algorithm by repeating the following steps: Randomly select a cycle, compute the cost of both orientations of the cycle, and include the cheaper orientation in the solution. He proved that if both orientations of every cycle of the graph results in same cost, his algorithm outputs an optimal solution. However, when applied to general graphs, the performance ratio of Guan's algorithm is unbounded as shown in Figure 4.2. The input graph is shown in Figure 4.2 (a). The output of Guan's algorithm (Figure 4.2 (b)) costs $1 + 5\epsilon$, whereas the optimal solution (Figure 4.2 (c)) costs 8ϵ. This example also shows that even if the input graph is of even degree, edges may have to be duplicated to attain the optimal WPP solution. Recall that the undirected CPP does not duplicate any edges for even-degree input graphs.

Win [51] proved that WPP is solvable in polynomial time for even-degree graphs. Win also presented a Linear Programming (LP) formulation to orient the edges and form directed tours and solved it using the minimum-cost flow

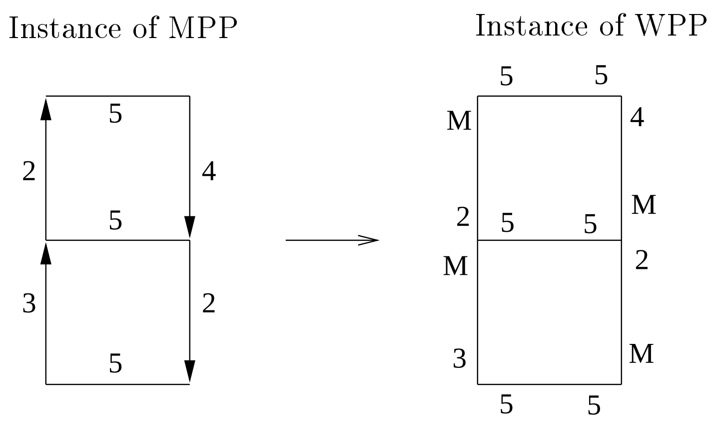


Figure 4.1: Reduction from MPP to WPP

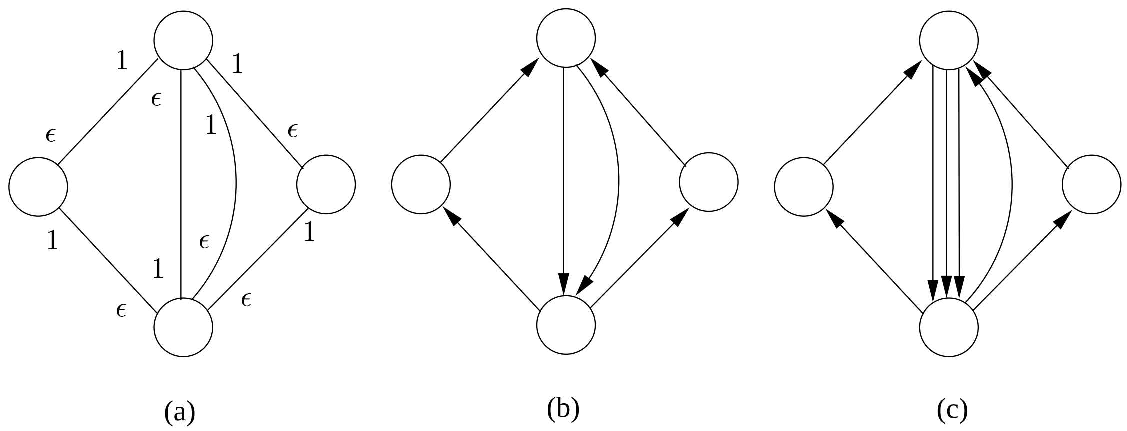


Figure 4.2: Guan's algorithm applied to general graphs

approach similar to the one presented by Edmonds and Johnson [11] for mixed graphs. He proved that the LP gives an integer solution for Eulerian graphs, and half-integral solution for general graphs. For general graphs, Win suggested to add a minimum-cost T-join of odd-degree nodes based on the average costs of the edges to make the graph Eulerian and then obtain an integral solution to the LP. Since a minimum-cost T-join duplicates each edge at most once, Win's algorithm has an approximation ratio of 2.

Several other contributions have appeared in literature about WPP, its LP formulation, and its polyhedral structure [24, 40, 50]. The reader is referred to one of several comprehensive surveys [4, 12, 52], for more references.

4.4 Our results

Other articles have been published in literature on WPP after Win's article in 1989, and yet his result has been the best approximation algorithm for WPP in terms of the proven worst case ratio till now. We present a new approximation algorithm for WPP combining the LP formulation for WPP and Frederickson's mixed strategy. We achieve an approximation ratio of 3/2 for WPP by observing critical properties of intermediate solutions, and proving an improved lower bound for the optimal solution.

In many instances, problems on graphs with asymmetric costs seem much harder than when the cost function is symmetric. Examples include the traveling salesman problem and the Steiner tree problem, among many others. In metric spaces, asymmetric versions of these problems seem to differ dramatically from their symmetric counterparts in terms of their approximability. The main obstacle seems to arise from the fact that given a cycle, the cost of traversing the edges of the cycle in one direction could be arbitrarily large compared to the cost of traversing the cycle in the opposite direction. Prior to our work, the best known approximation ratio for the symmetric and asymmetric versions of the postman problem were 3/2 and 2 respectively. We close this gap by developing an approximation algorithm for the windy postman problem with an approximation ratio of 3/2.

4.5 Linear program (LP) for WPP

Let $G = (V, E)$ be the input graph, and for each edge $(i, j) \in E$, let x_{ij} and x_{ji} be the usage of the edge from i to j and from j to i respectively. Then the LP formulation of the WPP is

$$\begin{aligned}
\min Z(G) &= \sum_{(i,j)\in E} (c_{ij}x_{ij} + c_{ji}x_{ji}) \\
\text{subject to} & \\
x_{ij} + x_{ji} &\geq 1,\ \forall (i,j) \in E \\
\sum_{j:(i,j)\in E} (x_{ij} - x_{ji}) &= 0,\ \forall i \in V \\
x_{ij}, x_{ji} &\geq 0\ (\forall (i,j) \in E)
\end{aligned}$$

In other words, this LP formulation minimizes the total traversal cost, while satisfying the following conditions:

- The total usage of each edge should be at least 1,
- The total incoming flow should be equal to the total outgoing flow at each node,
- The flow in both directions should be non-negative.

Intuitively, this LP creates circular flows in the graph by forming several directed cycles. The addition of each such cycle can be associated with a corresponding reduction in the cost of a solution. If additional constraints are added stating that all x_{ij}-values are integers, then the resulting integer program describes WPP completely.

Lemma 4.1 (Win [51]) *For an Eulerian input graph G, all (x_{ij}, x_{ji}) components of the vertices of the polyhedron P corresponding to the above LP are integral. For arbitrary graphs, the components of the vertices of the polyhedron P are half-integral.*

Win [51] has shown that the solution of a suitably designed flow problem yields such a solution to the LP. Without loss of generality, we assume that the optimal solution to the LP found by an algorithm is one of the vertices of the LP polyhedron structure formed by the constraints. If the input is an Eulerian graph, there are no undirected edges in the output graph. For general graphs, there can be undirected edges (corresponding to $x_{ij} = x_{ji} = 1/2$) present in the output, in addition to directed edges. Note that an optimal solution to the LP will duplicate and orient edges any number of times as long as it can form directed cycles which decrease the cost of the solution.

Let G be an arbitrary input graph to the LP, and let x^* be an optimal solution whose value is $Z^*(G)$. We construct a mixed multigraph $G_{LP} = (V, U, D)$ from x^* (with undirected edges U and arcs D) as follows: If $x_{ij} \geq 1$, then we place x_{ij} copies of the arc (i, j) in D; if $x_{ij} = 1/2$, then we add the undirected edge (i, j) to U. G_{LP}, a mixed graph, is an interpretation of the optimal solution, and it contains G as an underlying undirected graph. It can be verified that the cost of G_{LP} is $Z^*(G)$. Also, observe that $Z^*(G) \leq C(G)$, because $x_{ij} = x_{ji} = 1/2$ for all $(i,j) \in E$ is a feasible solution for the problem whose cost is $C(G)$.

4.6 Solving the LP

In this section, we will present an algorithm for solving the LP using a minimum-cost flow algorithm. This algorithm is similar to the one presented by Win to solve the LP on Eulerian windy graphs. We have modified it to solve the LP for general windy graphs. It is also similar to the algorithm presented by Brucker [4] to find an optimal INOUT augmentation for mixed graphs.

---

1. Input graph: $G = (V, E)$ with a cost function $c = (c_{ij}, c_{ji})$ for edges of E.
2. For each edge $(i,j) \in E$, if $c_{ij} \leq c_{ji}$, then orient the edge from i to j, else orient the edge from j to i. We obtain a digraph $D_G = (V, A)$.
3. Construct a digraph $D' = (V, A')$ with arc set A', arc costs, arc capacities and node demands defined as follows:
   (a) For each arc $(i,j) \in A$, define three arcs in A', namely, one copy of (i,j) with cost c_{ij} and infinite capacity, one copy of (j,i) with cost c_{ji} and infinite capacity, and one additional copy of (j,i), denoted by $(j,i)'$, with cost $(c_{ji} - c_{ij})/2$ and capacity 2. We will refer to such arcs as "artificial" arcs.
   (b) For each node $u \in V$, the demand is its deficit in D_G, i.e., $outdegree(u) - indegree(u)$. Note that a node with negative demand is considered as a supply node.

4. Find a minimum cost flow in D' satisfying the demands and supplies of all nodes. Let the output $y^* = (y^*_a)_{a \in A'}$ be the optimal flow, and y^*_{ij}, y^*_{ji}, and $y^*_{ji'}$ denote the flow values along the arcs $(i,j), (j,i) and (j,i)'$ respectively.

5. Construct the output graph $G_{LP} = (V, U, D)$ using the following rules:

   (a) If $y^*_{ji'} = 0$, then put $(y^*_{ij} + 1)$ copies of arcs in the output graph. y^*_{ji} will be 0 in this case.

   (b) If $y^*_{ji'} = 2$, then put $(y^*_{ji} + 1)$ copies of arcs in the output graph. y^*_{ij} will be 0 in this case.

   (c) If $y^*_{ji'} = 1$, then introduce an undirected edge (i,j) in the output graph. Both y^*_{ij} and y^*_{ji} will be 0 in this case.

Recall that the solution creates circular flows in the graph by forming several directed cycles. This algorithm comes up with such cycles by first directing the edges in the least cost direction, and then coming up with cycles using the minimum-cost flow. Note that, when the input graph has even degree, flow imbalances created in the first step at each node are even, and all the actual flow quantities can be kept even throughout the algorithm. So, $y^*_{ji'} = 1$ never occurs in the flow output (step 5(c)), hence there are no undirected edges in the solution. Therefore, this algorithm is an optimal algorithm for even-degree input graphs.

4.7 Overview of our algorithm

Our algorithm is similar in its overall structure to the algorithms for MPP presented in Chapter 3. But it differs significantly in some of the crucial steps.

Our algorithm finds two solutions for the windy postman problem and selects the solution of minimum-cost among the two. Both solutions are based on the half-integral optimal solution x^* to the LP.

The first solution S_1 is computed as follows. Consider the set of fractional edges:

$$U = \{(i,j) \in E \mid x^*_{ij} = 1/2\}.$$

Without loss of generality, we assume that the subgraph induced by the edges of U is acyclic, since any cycle can be oriented in one of its two directions without changing the cost of the solution. Since all x-values are either 0 or $1/2$ or a positive integer, $(i,j) \in U$ iff $(j,i) \in U$. It can be verified that the following solution, S_1, is an integral feasible solution for the WPP instance:

$$x'_{ij} = \begin{cases} 1 & \text{if } (i,j) \in U \\ x_{ij} & \text{if } (i,j) \notin U \end{cases}$$

In other words, S_1 contains all the edges oriented by the LP, and the duplicated "undirected" edges. Such duplicated edges can be oriented in opposite directions. Solution S_1 is preferable if the cost of the integral edges in the LP-solution is more than the cost of its fractional edges.

The second solution S_2 is obtained as follows. Suppose we can find an Eulerian subgraph S of G that contains all the edges U. Then we can take the two Eulerian orientations of S and "add" each of them to the LP solution, and then select the smaller of the two solutions as S_2. Given an integral solution (not necessarily feasible), we add another oriented tour to it as follows. For edges in U, we orient them in the direction of the Eulerian tour of S. For an edge $(i,j) \notin U$ which is used by the tour of S, if $x_{ij} > 0$ then increment x_{ij} by 1 (by adding another duplicate edge). Otherwise $x_{ji} > 0$, in which case, we decrement x_{ji} by

1. In order to make sure that the resulting solution is feasible, we have to avoid making $x_{ij} = x_{ji} = 0$ for any edge. Therefore, before adding an orientation of S to the LP solution, we double all integral x-values in the LP optimal solution.

The difficulty in making the above scheme work is that the graph G may not contain an Eulerian subgraph S that includes U. This happens when there is a cut that is crossed only by an odd number of edges of U. In this case, we are forced to duplicate some edges of U, and this cost has to be accounted. To handle this case, we prove an improved lower bound for the WPP by showing that the cost of duplicating a few edges of U can be accounted in the analysis. Solution S_2 is preferable when the cost of the fractional edges U in the LP-solution is more than the cost of the integral edges.

The algorithm returns the minimum-cost solution out of S_1 and S_2, and we show that its cost is at most $3/2$ times the cost of an integer optimal solution.

4.8 The algorithm

The approximation algorithm for WPP is given below:

---

1. Input graph: $G = (V, E)$
2. Solve the LP for WPP on G and let its output be $G_{LP} = (V, U, D)$. If it contains any undirected cycles, direct them arbitrarily. This does not affect the cost of the solution.
3. ALGORITHM1: Duplicate each edge of U and direct them in opposite directions. This forms the first solution S_1 for the problem.

4. ALGORITHM2:

   (a) Consider the subgraph of G_{LP} containing all its arcs and none of its edges. In that subgraph, each arc is part of one of the strongly connected components (SCCs). Shrink each such SCC in G_{LP} to a super node. For two super nodes A and B, set the cost of edge between A and B to the minimum cost of any edge (v, w) where $v \in A$ and $w \in B$. Find a minimum-cost T-join X that has odd degree at all odd degree super nodes and even degree at other super nodes. X contains a subset of U. Add X to G_{LP} and orient the two copies of each edge in opposite directions.

   (b) Now each SCC has even number of edges connecting it to the rest of the graph. Ignoring the direction of arcs, find a minimum-cost T-join of odd-degree nodes within each SCC.

   (c) Now, duplicate all the arcs within each SCC.

   (d) The union of the undirected edges and the T-join forms an Eulerian graph, and we add its Eulerian orientation to the solution as follows. Starting from an end-node of any undirected edge, walk on the edge and continue walking along the T-join. Whenever the walk goes along the orientation of the arc, add an additional copy of that arc. If the walk goes against the orientation of an arc, remove one of the copies of that arc. When the other end-node of the T-join is reached, walk along undirected edge to reach another odd-degree node, then walk along the T-join again as defined above. The walk ends when the initial vertex is reached, orienting all the undirected edges along the way. If undirected edges still exist after completing the walk, restart the walk from an end-node of one of these edges.

(e) Finally, all undirected edges become oriented. In this solution, all arcs of D that are part of the T-join (computed in step 4b), will have either one or three copies, and all other arcs of D will have two copies. Let us call this graph as S_3. Now re-do Step 4d, but walk along each cycle in the opposite direction. This will result in another solution S_4. Choose the solution with minimum cost between S_3 and S_4, and let us refer to it as S_2.

5. Output the solution with the lower cost between S_1 and S_2.

4.9 Structure of LP solutions

In this section, we identify a few critical properties of LP solutions. The ideas are similar to the ones used in Chapter 3. Let G be an input graph to WPP, and let G_b and G_r (blue graph and red graph respectively) be two LP solutions.

Define the operation of removing a set of arcs B from a LP solution G_{LP}, denoted by $G_{LP} - B$, as follows. If an arc (i, j) of B has $x_{ij} > 1$, then it can be removed. On the other hand, if $x_{ij} = 1$, then we remove the orientation by setting $x_{ij} = x_{ji} = 1/2$. The cost savings of such an operation is the sum of the directional costs of "removed" arcs and the orientation costs of other arcs:

$$\sum_{(i,j)\in B: x_{ij}>1} c_{ij} + \sum_{(i,j)\in B: x_{ij}=1} \frac{c_{ij} - c_{ji}}{2}.$$

Similarly, the addition operation $G_{LP} + B$, where a set of arcs B are added to G_{LP}, can be defined. The following Lemma is similar to Lemma 3.2 in Chapter 3 and can be proved similarly.

Lemma 4.2 *Consider two LP solutions x^r and x^b corresponding to graphs G_b and G_r respectively, and their symmetric difference G_{br}. Let AC be an alternating cycle in G_{br}, with blue arcs B and red arcs R. Then, $G_b - B + R$ and $G_r - R + B$ are also feasible solutions to the LP.*

Lemma 4.3 *Let $d^-(R)$ be the cost savings in removing R from G_r, i.e., $C(G_r) - C(G_r - R)$. Let $d^+(R)$ be the cost of adding R to G_b, i.e., $C(G_b + R) - C(G_b)$. Similarly, let $d^-(B)$ be the cost savings in removing B from G_b and let $d^+(B)$ be the cost of adding B to G_r. Then $d^+(R) \leq d^-(R)$ and $d^+(B) \leq d^-(B)$.*

Proof. We prove that the cost of adding R to G_b does not exceed the cost savings in removing R from G_r. Each arc (i, j) of R in G_r can be either additional oriented edge ($x^r_{ij} > 1$), or original oriented edge ($x^r_{ij} = 1$). Let us consider the contribution of (i, j) in both cases:

Case 1: $x^r_{ij} > 1$. Arc (i, j) is an additional oriented edge, so the contribution to $d^-(R)$ is its directional cost, c_{ij}. There are four cases to consider when adding arc (i, j) to G_b:

- $x^b_{ji} = 0$: Simply add additional arc (i, j) to G_b. Contribution to $d^+(B)$ is c_{ij}.

- $x^b_{ji} = 1/2$: (i, j) is an undirected edge in G_b. Orient the edge, its contribution to $d^+(B)$ is its orientation cost, $(c_{ij} - c_{ji})/2$.

- $x^b_{ji} = 1$: G_b has original oriented edge in the opposite direction. Remove the opposite orientation and leave it undirected. Its contribution to $d^+(B)$ is again its orientation cost.

- $x^b_{ji} > 1$: G_b has additional arcs in the opposite direction. Remove one arc. Its contribution to $d^+(B)$ is $-c_{ji}$.

It can be verified that $max(c_{ij}, (c_{ij} - c_{ji})/2, -c_{ji}) \leq c_{ij}$.

Case 2: $x^r_{ij} = 1$. Arc (i, j) is an original oriented edge in G_r, so its contribution to $d^-(R)$ is its orientation cost, $(c_{ij} - c_{ji})/2$. Since (i, j) is an original oriented edge in the symmetric difference G_{br}, G_b cannot have the same arc (i, j). So, among the four cases discussed above, case $x^b_{ji} = 0$ does not apply. In the other three cases, contribution of (i, j) to $d^+(R)$ does not exceed its contribution to $d^-(R)$. ∎

4.10 Improved lower bound for WPP

Consider an input graph $G = (V, E)$. Let C^* be the cost of an optimal postman tour of G. Suppose, the LP outputs $G_{LP} = (V, U, D)$. $U \subseteq E$ are the edges of G that were not oriented by LP. D consists of arcs that satisfy $indegree = outdegree$ at each vertex. Note that $C(D) + C(U) = Z^*(G)$. The subgraph induced by D may contain several strongly connected components (SCC's). Recall that in a SCC, there exists a directed path between every pair of nodes in the component. Suppose we shrink each SCC into a single vertex and form an undirected graph UG. Recall that the cost of an edge is the average of its directional costs. Adding a minimum-cost T-join of odd degree nodes on UG, a subset of U, will make the degree of each vertex in UG even. Let us refer to the T-join edges as X. Recall the following lemma from Chapter 3, and then we show the improved lower bound on C^*.

Lemma 4.4 *Let $G = (V, E)$ be the input graph. Let $G_{LP} = (V, U, D)$ be an optimal LP solution. Let $U = \{u_1, ..., u_k\}$ be the undirected edges of G that were not oriented by LP. Define a new input graph H by adding n_i copies of u_i $(i = 1, \ldots, k)$ to G. Let Y be the set of such additional edges. We claim that $Z^*(H) = Z^*(G) + C(Y)$.*

Lemma 4.5 $Z^*(G) + C(X) \leq C^*$.

Proof. Let G^* be an optimal solution for WPP, whose cost is C^*. Consider the super nodes of UG, the nodes corresponding to the strongly connected components of D. G^* needs to have additional T-join edges (a subset of U) between these components to satisfy the even-degree condition for each component. Let us denote these T-join edges by X'. We know that $C(X')$ is at least $C(X)$ since X is a minimum-cost T-join. By Lemma 4.4, we know that adding X' to input graph G will result in an optimal LP solution whose cost is $Z^*(G) + C(X')$, which is at least $Z^*(G) + C(X)$. Since the LP is a relaxation of the WPP, the cost of the optimal WPP tour is at least $Z^*(G) + C(X)$. ∎

4.11 Analysis

The algorithm first runs the LP on input graph $G = (V, E)$ and obtains $G_{LP} = (V, U, D)$ with cost $Z^*(G) = C(D) + C(U)$.

Lemma 4.6 ALGORITHM1 *outputs a solution that costs no more than $2C^* - C(D)$.*

Proof. ALGORITHM1 duplicates all edges of U, so the cost of solution returned by ALGORITHM1 is $C(D) + 2C(U)$. Since $C(D) + C(U) = Z^*(G) \leq C^*$, we get $C(S_1) \leq 2C^* - C(D)$. ∎

Lemma 4.7 ALGORITHM2 *outputs a solution that costs no more than* $C^* + C(D)$.

Proof. After adding a minimum-cost T-join of odd-degree SCC's (X) and doubling SCC's (D), the cost of the intermediate graph is $C(U) + 2C(D) + C(X) = Z^*(G) + C(D) + C(X)$. This is at most $C^* + C(D)$ by Lemma 4.5. Now, consider the two directed walks computed in Steps 4(d) and 4(e). If one solution adds a particular arc, the other solution deletes that arc. Also, both solutions orient undirected edges in opposite directions. Therefore the sum of the cost of both solutions is at most $2C^* + 2C(D)$. Hence the smaller of the two solutions is at most $C^* + C(D)$. ∎

Theorem 4.1 *The algorithm in Section 4.8 finds a solution of cost at most* $1.5\ C^*$ *to the windy postman problem.*

Proof. When $C(D) \geq C^*/2$, by Lemma 4.6, ALGORITHM1 outputs a solution with cost at most $2C^* - C(D)$, which is at most $1.5\ C^*$. When $C(D) \leq C^*/2$, by Lemma 4.7, ALGORITHM2 outputs a solution with cost at most $C^* + C(D)$, which is less than or equal to $1.5\ C^*$. Since, our algorithm chooses the solution with lower cost in Step 5, the cost of the final solution is at most $1.5\ C^*$. ∎

4.12 Example

Figure 4.3 (a) specifies an input graph for WPP. The directional costs of edges are specified near the tail end of the corresponding arcs. After running the LP in Step 2, we get the graph in Figure 4.3 (b), which contains two SCC's. ALGORITHM1 duplicates all the undirected edges and orients them in opposite directions, and we get solution S_1 (Figure 4.3 (c).) As part of ALGORITHM2, we compute a minimum-cost T-join and add it to G_{LP} (Figure 4.4 (a)). We duplicate all the SCC's in Figure 4.4 (b). Now the algorithm computes a T-join within each SCC. Adding two orientations of the T-join, we get S_3 (Figure 4.4 (c)) and S_4 (Figure 4.4 (d)) respectively. The minimum cost solution among Figure 4.3 (c), Figure 4.4 (c), and Figure 4.4 (d) is Figure 4.4 (d), with cost 50.

4.13 Minimum cost Eulerian orientation problem (MCEOP)

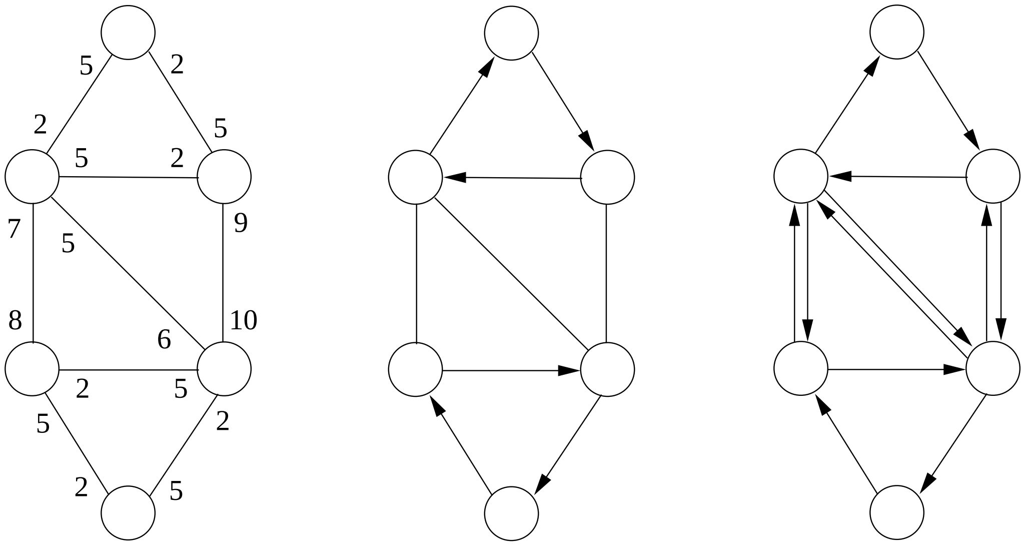
A combinatorial optimization problem that is closed related to the WPP on Eulerian Graphs is the minimum-cost Eulerian orientation problem: Given a connected, even-degree graph and an asymmetric cost function C, the goal is to find a minimum-cost Eulerian orientation of the graph, without duplicating any edges. Guan and Pulleyblank [27] presented the first algorithm for this problem. Win [51] presented a simple algorithm for this problem by making the following change to the algorithm for WPP: in the minimum-cost flow formulation, limit the capacity of each arc to 1 instead of infinity.

4.14 Capacitated WPP

This problem is another extension of WPP: In addition to the usual WPP requirements, two integers (capacities) d_{ij} and d_{ji} are specified for each edge (i, j).

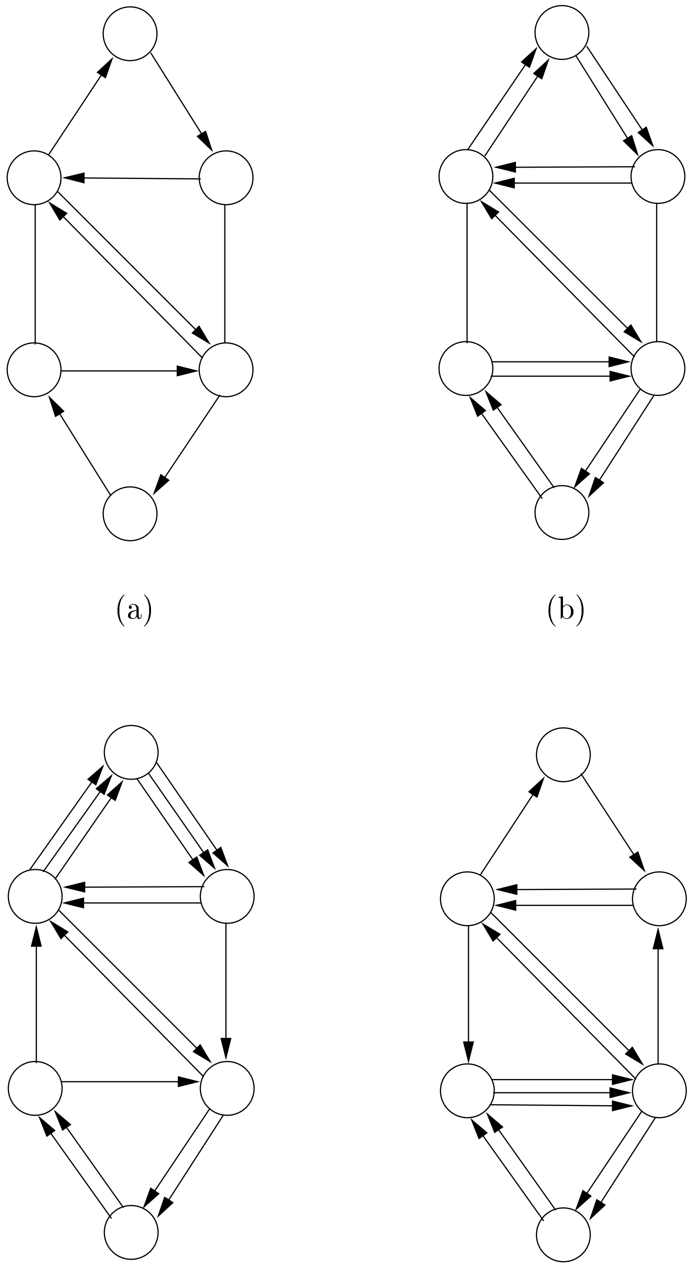
The goal is to find a minimum-cost windy postman tour obeying the capacity constraints, and additionally the total usage of each edge should be at most $d_{ij} + d_{ji} - 1$. When $d_{ij} = d_{ji} = 1$ for each edge, this problem becomes the minimum cost Eulerian orientation problem (MCEOP). Win proved that, if d_{ij} and d_{ji} are odd for every edge $(i, j) \in E$, a polynomial-time algorithm can be designed using minimum-cost flow approach. He showed that the following constraints form a complete description of the polytope for this problem:

$$\begin{aligned}
\min Z(G) &= \sum_{(i,j)\in E} (c_{ij}x_{ij} + c_{ji}x_{ji}) \\
\text{subject to} & \\
1 \leq x_{ij} + x_{ji} &\leq d_{ij} + d_{ji} - 1, \ \forall (i,j) \in E \\
\sum_{j:(i,j)\in E} (x_{ij} - x_{ji}) &= 0, \ \forall i \in V \\
x_{ij}, x_{ji} &\geq 0, \ \forall (i,j) \in E
\end{aligned}$$



(a) (b) (c)

Figure 4.3: An example for WPP algorithm: Part 1



(a) (b)

(c) (d)

Figure 4.4: An example for WPP algorithm: Part 2

CHAPTER 5

MPP - RESTRICTED VERSIONS

The mixed postman problem (MPP) is usually solved in two steps: First, edges and arcs are added to the input graph to make it satisfy the necessary conditions for the existence of an Euler tour, and then an actual tour is found. During the first step, any edge or arc may be duplicated any number of times. An interesting variation of MPP is when this duplication is limited to a specified subset of edges and arcs. Since MPP is NP-hard, this restricted version of MPP is also NP-hard. MPP with such additional constraints was first suggested by Edmonds and Johnson [11]. We study the following special cases: First, we consider the restricted MPP on even-degree input graphs and present a polynomial time algorithm to solve the problem. Then, we consider all mixed graphs, but limit duplication to edges only, and present a 1.5-approximation algorithm for this version. Finally, we will study the restricted MPP with duplication restricted to arcs only.

5.1 Restricted MPP

The problem statement is as follows: given a mixed graph $G = (V, E, A)$ with cost function C on $E \cup A$, a subset of edges $E' \subseteq E$, and a subset of arcs $A' \subseteq A$, the goal is to find a minimum-cost postman tour of G in which duplication is restricted to $E' \cup A'$, i.e., the remaining edges and arcs should occur exactly once in the output tour. When $E' = E$ and $A' = A$, the restricted MPP becomes the classical MPP problem discussed in Chapter 3. Since the classical MPP is a subset of the restricted MPP, the NP-hardness of classical MPP implies that the restricted MPP is also NP-hard. In the restricted MPP, the specified subset of edges E' and arcs A' are responsible for correcting the imbalances at the nodes.

Even though the restricted MPP is NP-hard, there exists a polynomial time algorithm if the input is limited to even-degree graphs. Recall that the algorithms for MPP augmented the input mixed graph to satisfy the sufficient conditions: Even degree condition and symmetry condition (see Chapter 3). In restricted MPP on even-degree graphs, the even degree condition is already satisfied. We will present an algorithm DUP_SUBSET that uses minimum-cost flows to find an optimal solution. This algorithm is derived from similar algorithms presented by Edmonds and Johnson [11], Brucker [4], and Win [51]. The core idea of DUP_SUBSET is that the flow is restricted to use the specified subset of edges and arcs as many times as needed, but is limited to use all other edges and arcs exactly once.

5.1.1 Algorithm DUP_SUBSET

---

1. Input: A mixed graph $G = (V, E, A)$, a cost function C on $E \cup A$, a subset of edges $E' \subseteq E$, and a subset of arcs $A' \subseteq A$.

2. Orient each edge of G arbitrarily to obtain the digraph $D_G = (V, B)$, Note

that D_G has the arcs of G and in addition has one arc for each edge of G.

3. Construct a flow network $D' = (V, B')$ with arc set B', arc costs, arc capacities and node demands computed from G and D_G as follows:

   (a) For each arc $(i, j) \in A'$, add a copy of (i, j) to B' with cost c_{ij} and infinite capacity.

   (b) For each oriented edge $(i, j) \in B$, add a copy of $(j, i)'$ with cost 0 and capacity 2. We will refer to such arcs as "artificial" arcs. If the corresponding edge $(i, j) \in E'$, introduce two more arcs in B': One copy of (i, j) and one copy of (j, i) each with cost c_{ij} and infinite capacity.

   (c) For each node $u \in V$, the demand is its deficit in D_G, i.e., $outdegree(u) - indegree(u)$. Note that negative demand is considered as the supply.

4. Find a minimum-cost flow in D' satisfying the demand/supply at each node without violating the capacities of arcs. Let its output be $y^* = (y_a^*)_{a \in B'}$ be an optimal flow, and let y_{ij}^*, y_{ji}^*, and $y_{ji'}^*$ denote the flow values along the arcs (i, j), (j, i) and $(j, i)'$ respectively.

5. Construct the output digraph using the following rules:

   (a) For each arc (i, j) in $A - A'$, introduce one copy of arc (i, j) in the output graph.

   (b) For each arc (i, j) in A', introduce $(y_{ij}^* + 1)$ copies of (i, j) in the output graph.

(c) For each edge (i, j) in $E - E'$, if $y^*_{ji'} = 0$, then introduce one copy of oriented edge (i, j) in the output graph, otherwise introduce one copy of oriented edge (j, i) in the output graph.

(d) For each edge (i, j) in E', if $y^*_{ji'} = 0$, then introduce $(y^*_{ij} + 1)$ copies of oriented edge (i, j) in the output graph, otherwise introduce $(y^*_{ji} + 1)$ copies of oriented edge (j, i) in the output graph.

Theorem 5.1 *Algorithm* DUP_SUBSET *outputs an even-degree graph that satisfies the symmetry condition.*

Proof. The supply/demand for each node in digraph D_G is even, since G is an even-degree graph. The capacity of all arcs in D' is either 2 or infinity. The minimum-cost flow algorithm guarantees an integer solution to any flow problem with integer inputs. Additionally, since all capacities are even, there is no loss of generality in assuming that all flow values will be even.

The edges are arbitrarily oriented to get D_G. The purpose of artificial arcs is to reverse the direction of each edge, if required, by the flow. For each oriented edge (i, j) in D_G, adding an artificial arc $(j, i)'$ with cost 0 and capacity 2 effectively enables the flow algorithm to reverse the direction of flow through that edge, without affecting the parity of adjacent nodes. Other y^*_a values that correspond to the additional arcs/oriented edges in the output graph do not change the degree of the nodes. So, the output is an even-degree graph. Additionally, it satisfies the symmetry condition, since the flow has annulled the surplus/deficit imbalance. ∎

Since the algorithm used minimum-cost flow to duplicate edges and arcs, the algorithm DUP_SUBSET is an optimal algorithm for the restricted MPP on even-degree graphs. This algorithm also can be used to find whether the restricted MPP on a particular even-degree graph is feasible. If an instance is not feasible, the corresponding flow problem will fail to satisfy all the demands. If it is feasible, the algorithm outputs a digraph which satisfies the requirements of restricted MPP. The actual Euler tour on the digraph can be found using algorithms presented in Chapter 3.

Figure 5.1 (a) contains an even-degree input graph G in which only edges and arcs marked by dashed lines may be duplicated. Figure 5.2 shows the minimum-cost flow formulation steps for edge (a, b). The edge costs 5 and it is directed from a to b in D_G. Correspondingly, the following three arcs are defined in the flow graph D': arc (a, b) with cost 5 and infinite capacity, arc (b, a) with cost 5 and infinite capacity, arc $(b, a)'$ with cost 0 and capacity 2. Figure 5.1 (b) shows the output of algorithm DUP_SUBSET. The classical MPP solution for the input graph is shown in Figure 5.1 (c). Note that, due to the additional restrictions, the restricted MPP solution costs more than the classical MPP solution.

Algorithms similar to MIXED1 cannot be applied directly to the restricted MPP on general mixed graphs. Algorithm EVENDEGREE may duplicate some arcs, and the resulting graph may become infeasible. Figure 5.3 shows such an example. Figure 5.3 (a) shows a specific cut $(S, V - S)$ which is crossed by two edges and two arcs. Suppose these arcs belong to A' but the edges do not belong to E' and therefore cannot be duplicated and further suppose there are exactly two odd degree nodes in each of S and $V - S$. In trying to make all nodes even

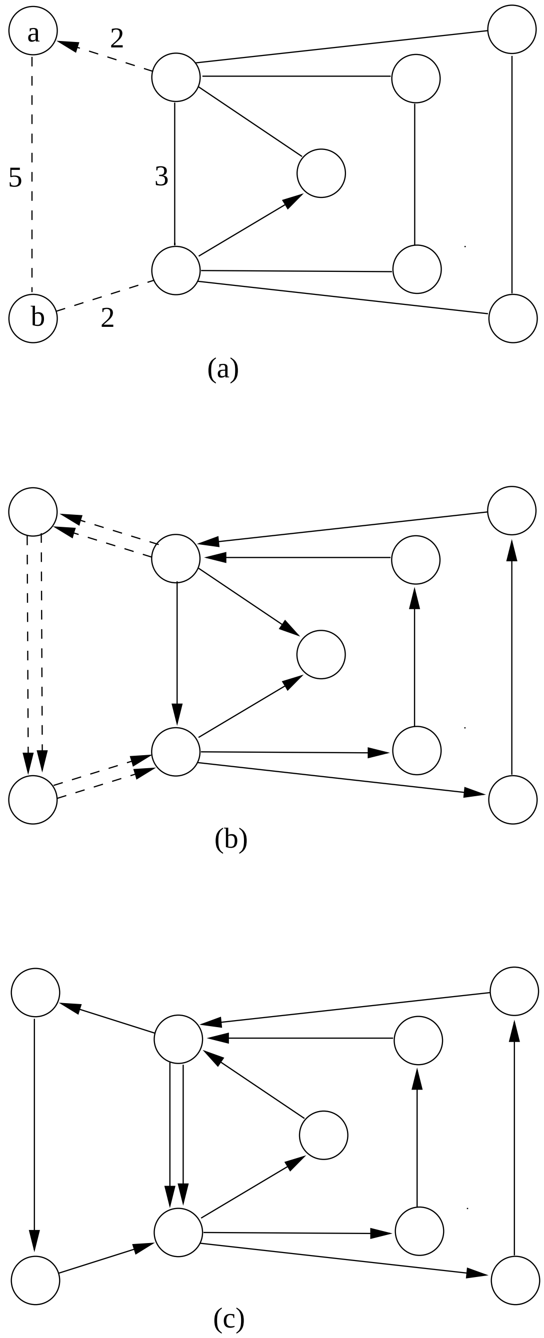


Figure 5.1: Restricted MPP on even-degree graphs

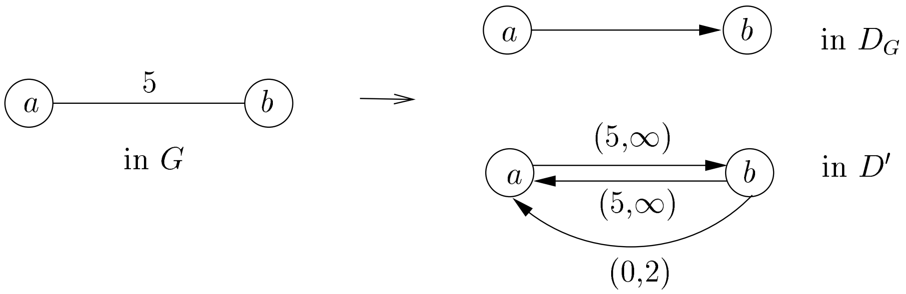


Figure 5.2: Minimum cost flow formulation

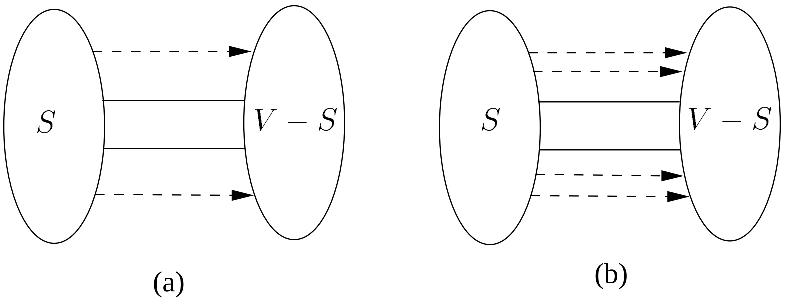


Figure 5.3: Algorithm MIXED1 applied to restricted MPP

degree, algorithm MIXED1 may duplicate that arcs making the resulting graph infeasible (Figure 5.3 (b)), since the edges cannot be duplicated to compensate the imbalance. On the other hand, duplicating any edge in E' can not make the graph infeasible, since it can be duplicated again and oriented in the opposite direction to nullify the effect. One way to analyze the problem is to run EVENDEGREE such that it will not affect the feasibility of restricted MPP in that graph. Consider an instance of a restricted MPP. Consider the subgraph $G' = (V, E', A')$. Construct another digraph G'' by copying all arcs, and introducing two arcs in opposite directions for each edge in E'. Now consider the strongly connected components of G''. If each strongly connected component has even number of odd-degree nodes of G, EVENDEGREE algorithm can be run without affecting the feasibility. Then algorithm DUP_SUBSET can be applied to find the solution. This provides a 2-approximation algorithm.

5.2 MPP - duplicate edges only

Given a mixed graph $G = (V, E, A)$ with cost function c on $E \cup A$, in which E is a set of edges, and A is a set of arcs, the goal is to find a minimum-cost postman tour in which only edges of E may be traversed more than once, i.e., restricting the duplication to edges only. Since arcs cannot be duplicated, we can reformulate the problem as follows: Find the surplus/deficit of each node based on the arcs, and assign it as the supply/demand of that node. Now find a minimum-cost flow in the graph $G' = (V, E)$, which balances the supply/demand of the nodes and also uses all edges at least once.

Recall the necessary and sufficient conditions for the existence of an Euler tour in a directed graph: The graph should be connected and be symmetric, i.e., $indegree(S) = outdegree(S)$ for any cut $(S, V - S)$. Consider a cut that is crossed only by arcs in this problem: Since none of the arcs can be duplicated, such cuts should already satisfy the symmetry condition, otherwise the problem is infeasible. When a graph contains one or more cuts that are crossed only by arcs, then the undirected subgraph (V, E) is not connected, and it has two or more components. However, the symmetry condition guarantees that the total demand/supply of each component is 0. Figure 5.4 (a) shows a cut which contains only arcs and the cut is not balanced. Duplication of edges alone cannot make the graph symmetric. Figure 5.4 (b) shows the undirected components of the graph. Note that the net supply of the upper component is 1 and the net demand of the lower component is 1. This imbalance cannot be corrected since there is no additional flow possible between these components.

Therefore, solving MPP with duplication restricted to edges is equivalent to to finding a minimum-cost flow to satisfy the demand/supply requirements in each of the connected components of $G' = (V, E)$. We conjecture that the MPP with duplication restricted to edges is NP-hard and prove that the following algorithm DUP_EDGES is a 1.5-approximation algorithm for this problem.

5.3 Algorithm DUP_EDGES

Find the components of $G' = (V, E)$, and run the algorithm DUP_EDGES on each component.

- Input: an undirected, connected graph $H = (V, E)$ with cost function c on edges, and demand d on nodes. Note that c can be generalized to a set of edges by taking the sum of their costs, and d can be generalized to a set of nodes by taking the sum of their demands.

- Find the bridge edges of H. Find the imbalance across each bridge, i.e., if a bridge edge (i, j) connects sub-components H_i and H_j, find the total demand of all the nodes in H_i. Note that $d(H_i) = -d(H_j)$.

- For each bridge edge (i, j), let $d(H_i) = m$ and $d(H_j) = -m$. If $m > 0$, introduce m copies of (i, j), and orient all of them from j to i to satisfy the symmetry conditions, and adjust $d(i)$ and $d(j)$ accordingly. If $m = 0$, duplicate the edge exactly once to satisfy the parity conditions, and orient the original and duplicate edges in opposite directions. In this case, $d(i)$ and $d(j)$ remain unchanged.

- The previous step may create new components, since the processing of each cut adds one more component. However, the net demand/supply of each component still remains zero. Additionally, there are no more bridge edges, and hence each remaining connected component is 2-edge connected. For each such component H_i, execute the following steps:

  – Run the minimum-cost flow algorithm to satisfy the supply at each node in H_i. The flow algorithm may orient or duplicate and orient edges in the process. It outputs a symmetric, mixed graph H_i'.

  – Output graph may still contain a few unused edges. Without loss of generality, we can assume there are no undirected cycles, since we can orient any cycle arbitrarily to create a 0-cost circulation.

  – Now find the odd-degree nodes of H_i' considering only the remaining edges. Find a minimum-cost T-join of those nodes within the component. Once these edges are added to H_i', all nodes have even degree, i.e., one or more undirected cycles can be formed and arbitrarily oriented. The resulting graph is completely oriented, and is symmetric.

Combining the outputs for all components and the arcs in the input graph, we get a connected, symmetric digraph. Algorithms discussed in Chapter 3 can be used to find an actual Euler tour in this graph.

Lemma 5.1 *Algorithm* DUP_EDGES *finds a solution of cost at most* $1.5C^*$ *where* C^* *is the cost of optimal solution to the MPP with duplication restricted to edges.*

Proof. Consider the steps of the algorithm DUP_EDGES. The optimal solution also has to pay for all steps except the last one. In the last step, we add a T-join within each 2-connected component. Frederickson and JáJá [22] proved that the cost of a minimum-cost T-join in a biconnected, weighted graph is at most half the cost of the graph. It can be shown that the above statement is also true for any 2-edge connected graphs. Since C^* is at least $C(G)$, the weight of the T-join is at most $C^*/2$. Hence, the cost of the solution computed by the above algorithm is at most $1.5C^*$. ∎

Figure 5.5 shows an example for algorithm DUP_EDGES. Figure 5.5 (a) shows the input graph with node demands. The edge costs are not shown for simplicity. The graph has one bridge edge, and the upper component has a net demand 2, and the lower component has a net demand of -2. So the algorithm duplicates and orients the bridge edge to balance the demands as shown in Figure 5.5 (b). The rest of the algorithm is run within each component separately: A minimum-cost flow algorithm is run within each component to balance the node demands. The output of the flow is shown in Figure 5.5 (c). Now, the upper component still has an undirected edge, and the lower component has two undirected edges. Each component also has two odd-degree nodes each. A mininum-cost T-join of these odd-degree nodes is computed within each component. The graph obtained after adding the T-join edges is shown in Figure 5.5 (d).

5.4 MPP - duplicate arcs only

Given a mixed graph $G = (V, E, A)$ with cost function c on $E \cup A$, in which E is a set of edges, and A is a set of arcs, the goal is to find an Eulerian tour with restriction on duplication to arcs only. Unlike MPP with restriction on duplication to edges, this problem does not lend itself to a simple solution. In the NP-hardness proof given by Papadimitriou [39], the gadget utilized only duplication of arcs. Hence, the same proof shows that the MPP with duplication restricted to arcs is NP-hard.

The following conditions are necessary but not sufficient for the MPP with duplication restricted on arcs: Any cut that is crossed by only edges should contain an even number of edges so that half of them can be directed one way, and the other half can be directed the other way. An infeasible graph is shown in Figure 5.6 (a). In any cut that contains edges but has all arcs going in the same direction, the number of edges should be equal to or more than the number of arcs, since edges have to carry the flow in the reverse direction. Figure 5.6 (b) shows an infeasible case which violates this condition. However, Figure 5.6 (c) shows that these conditions are not sufficient.

We can run the minimum-cost flow to balance the input graph, with the restriction that the maximum edge capacity is 1. If the minimum-cost flow algorithm fails to balance the graph, then clearly the input graph does not have a solution. On the other hand, even if the balancing succeeds, as the example shows, there may not be a solution. The main obstacle seems to be the non-availability of additional directed paths, since the edges have been oriented by the flow.

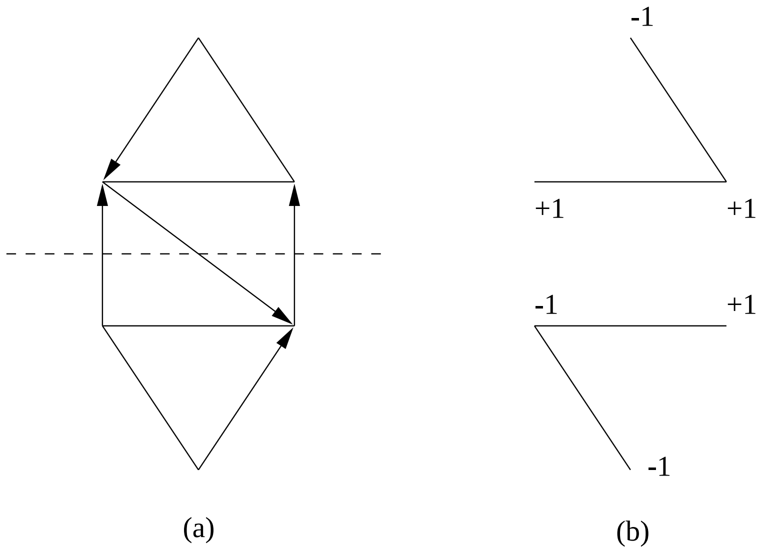


Figure 5.4: MPP with duplication restricted to edges: An infeasible case

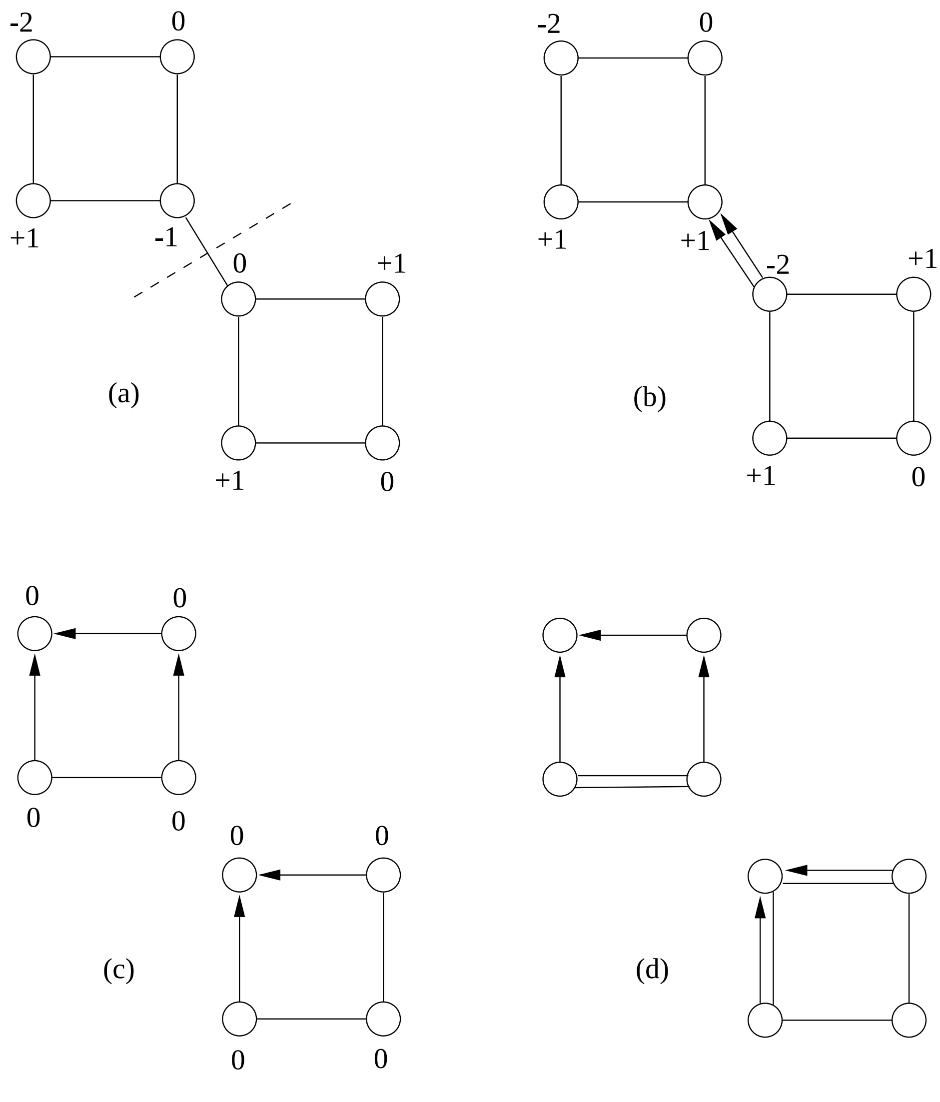


Figure 5.5: Algorithm DUP_EDGES

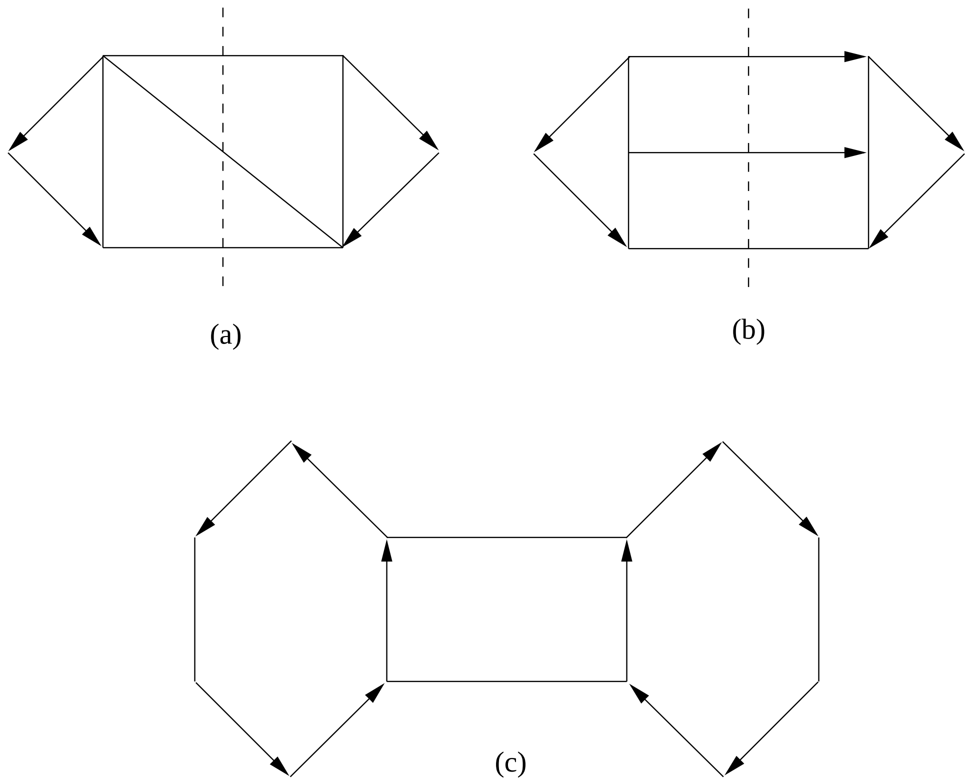


Figure 5.6: MPP with duplication restricted to arcs: A few infeasible cases

CHAPTER 6

EULER TOUR WITH RESTRICTED TURNS

Well known applications of the postman problem are mail delivery, garbage collection, parcel delivery, school bus routing, snow plowing, etc. Several algorithms to solve various versions of postman problem have been presented. Postman problems are solved in two stages. The first stage duplicates the necessary edges/arcs to satisfy the sufficient conditions for the existence of a postman tour, and the second stage applies an algorithm to find an actual tour. Consider the following practical constraint. Not all turns are permitted at each intersection. Additionally, the vehicles used may have large size. Even when allowed by traffic rules, these vehicles cannot make U-turns in many intersections. Even though left turns may be easier than U-turns, they can be difficult and dangerous in heavy traffic. So, the quality of an Euler tour can be associated with the ease and safety of the turns that are involved. So, the second stage of the postman problem should find an Euler tour with the best quality among all the tours possible. Two versions of Euler tour problems are studied in this chapter.

6.1 Directed Euler tour with restricted turns

In this version, we do not allow those turns that are not acceptable. Consider a connected, symmetric, directed graph $G = (V, A)$. Once some turns are disallowed, can we find a directed Euler tour that uses only valid turns? These restricted turns can be specified in various ways. We will use an $|A| \times |A|$ boolean matrix. A 0-entry specifies that the turn is not allowed and a 1-entry specifies that the turn is allowed. Similar to the traditional Euler tour problem, we should be able to start the tour on any arc and traverse all other arcs before reaching the first arc again. So, the starting node for the Euler tour is not significant.

We show that the directed Euler tour with restricted turns is NP-complete by reducing the unweighted directed traveling salesman problem (also known as the directed Hamiltonian cycle problem) to this problem. Given an instance $G = (V, A)$ of the directed TSP problem, construct an instance of directed Euler tour with restricted turns $H = (u, B)$ by following these steps:

- For each node i in G, introduce a self-loop arc $e_i = (u, u)$ in B.
- Initialize that all turns are not allowed.
- For each arc (i, j) in G, allow the turn from e_i to e_j.

Remark: The graph can be made simple by introducing a new vertex in the middle of each arc e_i, $i = 1, \cdots, |V|$.

Lemma 6.1 *There exists a solution for the unweighted directed TSP on G if and only if there is a solution for the directed Euler tour with restricted turns in H.*

Proof. Let $w = (i_1, i_2, \cdots, i_n, i_1)$ be a directed TSP tour in G. The TSP tour starts at i_1, visits all other nodes, and returns back to i_1. Construct an edge list by mapping each node to corresponding arc. Let $w' = (e_{i_1}, e_{i_2}, \cdots, e_{i_n}, e_{i_1})$ be that list. Since w is a valid TSP tour, there exists an arc from i_j to i_{j+1} in G. Correspondingly, the turn from e_{i_j} to $e_{i_{j+1}}$ is valid in H. So, $w'' = (e_{i_1}, e_{i_2}, \cdots, e_{i_n})$ traverses all arcs exactly once using only the allowed turns, and the presence of e_{i_1} after e_{i_n} in w' signifies that the Euler tour can restart in e_{i_1} since e_{i_n} to e_{i_1} is a valid turn. Hence, w'' is a valid Euler tour with restricted turns.

Let us prove the other direction now. Let $W = (e_{j_1}, e_{j_2}, \cdots, e_{j_n})$ be a valid Euler tour with restricted turns in H. By its definition, its turns for e_{j_k} to $e_{j_{k+1}}, k = 1, 2, \cdots, n-1$, and the turn (e_{j_n}, e_{j_1}) are valid turns. Find the corresponding node list in $W' = (j_1, j_2, \cdots, j_n)$ from w'. Since each turn e_{j_k} to $e_{j_{k+1}}$ is valid, there exists an arc from each j_k to j_{k+1} in G. Additionally, there exists an arc from j_n to j_1 in G, since (e_{j_n}, e_{j_1}) is a valid turn in H. So the node list $W'' = (j_1, j_2, \cdots, j_n, j_1)$ is a valid TSP tour. ∎

Theorem 6.1 *Directed Euler tour with restricted turns is NP-complete.*

Proof. Lemma 6.1 proved that the directed TSP can be reduced to the directed Euler tour with restricted turns. Since the directed TSP is NP-hard, the directed Euler tour with restricted turns is also NP-hard.

Given a instance of the directed Euler tour with restricted turns and a solution, it is straight-forward to verify whether it is a valid solution: If the given solution is $(e_1, e_2, \ldots . e_m)$, verify whether each turn e_i to e_{i+1} is valid. Additionally, verify whether the turn from e_m to e_1 is valid. If all these turns are valid, the given solution is a valid solution. So, a given solution can be verified in polynomial time. So the directed Euler tour with restricted turns is in NP. Since this problem is in NP and also NP-hard, it is NP-complete. ∎

6.2 Directed Euler tour with weighted turns

If the directed Euler tour with restricted turns does not exist for a directed graph, at least such undesirable turns can be kept to a minimum. So, instead of specifying allowable turns, we can capture the level of difficulty of a turn by attaching a weight with each turn. In fact, even the seemingly impossible U-turn can be assigned a weight, since it can represent the cost of a reasonable detour which is equivalent to the U-turn in effect. Given a connected, symmetric, directed graph $G = (V, A)$ with a cost function for arcs and turns, the problem is to find a minimum-cost tour visiting all arcs at least once. Note that the total cost is the sum of the arc-costs and the costs of the turns made by the tour. This problem is also NP-hard, since it is a generalization of the directed Euler tour with restricted turns.

Roy and Rousseau [45] considered the directed postman problem with weighted turns and suggested a mapping from the input graph by duplicating vertices and introducing additional arcs with the costs of corresponding turns (Figure 6.1). There is one dashed arc for each valid turn at the intersection. Also the cost associated with each turn is assigned to the corresponding arc. The output becomes an instance of the directed rural postman problem, which is the problem of finding a minimum-cost directed tour that uses all the solid arcs at least once, utilizing the necessary dotted arcs in the process.

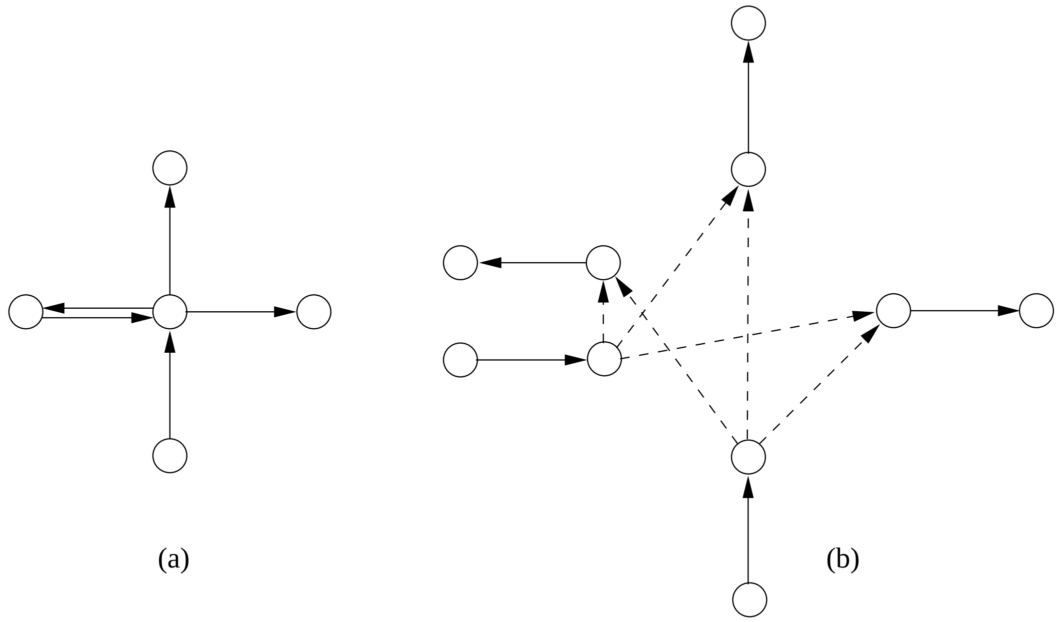


Figure 6.1: Directed postman problem with weighted turns

CHAPTER 7

RURAL POSTMAN PROBLEM

In the traditional CPP, all edges have to be traversed at least once. In several practical situations, not all edges need to be traversed. In these cases, a subset of edges are specified in the input as those edges that need to be visited by the postman tour. The goal is to find a postman tour traversing all the edges in the specified subset at least once. Such a tour can make use of the remaining edges and arcs, if needed. For example, if a postman have to deliver mail in rural areas, he needs to cover all the streets within different villages, but not all the roads connecting them. Hence the name of the problem is rural postman problem (RPP). Another application of RPP is school-bus routing. The school bus is required to traverse all streets in the residential areas. Roads in non-residential areas and other parts of the city need not be traversed.

7.1 Undirected RPP

Given a connected, undirected graph $G = (V, E)$ and $R \subseteq E$ with cost function c on edges, the objective is to find a minimum-cost tour that visits each edge of R at least once. Lenstra and Rinnooy Kan [32] proved that undirected RPP is NP-complete by providing a reduction from undirected Hamiltonian circuit problem.

Consider the subgraph $G_R = (V, R)$. Orloff [38] presented an optimal algorithm for RPP if G_R is connected, i.e., the edges of R form a connected subgraph in G. His algorithm is similar to that of undirected CPP: Identify the odd-degree nodes in G_R, find a minimum-cost T-join of these nodes in G, and add the T-join edges to G_R to form the solution graph. The solution graph is Eulerian since it is connected and has even degree. If G_R is not connected, this algorithm may come up with disconnected Euler subtours. Orloff [38] presented the following algorithm for the general case:

- Construct a graph $G' = (V', E')$ where V' contains all the nodes incident to R, and E' includes R and the edges between every pair of nodes in R with the corresponding shortest path costs in G.
- Construct a minimum cost forest in G' to connect all the connected components of G_R. Let the additional edges be T.
- Determine the odd-degree nodes of $(V', R \cup T)$, and find a minimum-cost T-join of these nodes in G'. Let the T-join edges be M.
- $(V', R \cup T \cup M)$ is a connected, even-degree graph. Replace each edge (i, j) of $T \cup M$ by the edges of a shortest path between the nodes i and j in G. Find an Euler tour in the output graph.

Since the optimal solution also has to connect G_R, $C(R+T) \leq C^*$. Additionally, the T-join edges may cost at most $C^*/2$, since triangle inequality is satisfied by G'. Therefore, the cost of the solution graph $C(R+T+M) \leq 1.5C^*$.

A sample execution of the algorithm for the undirected RPP is shown in Figure 7.1. Figure 7.1 (a) shows the input graph G in which the solid edges are the edges that have to be traversed at least once, and the dashed lines are the additional edges that can be used if needed. Figure 7.1 (b) shows G'. The minimum spanning tree edges are computed and added to connect the components in Figure 7.1 (c). A minimum-cost T-join of odd degree nodes is computed to make the graph have even degree (Figure 7.1 (d)), so an Euler tour can be found in the output graph.

As noted by Frederickson [20], the determining factor in the complexity of undirected RPP is the number of disconnected components in R. It is possible to create an exact recursive algorithm that is exponential only in the number of disconnected components.

Two integer programming formulations have been used to find exact solution for the undirected RPP. Christofides [7] presented a formulation using the following constraints: The degree of every node should be even, and any cut that divides the nodes of R should have at least 2 edges crossing them in the solution. The solution output by the above approximation algorithm was used as the initial solution in a branch and bound algorithm implementing this integer program. Corberan and Sanchis [9] studied a similar IP formulation specifying almost the same constraints, but came up with a different solution strategy identifying a number of families of facets of the polytope of feasible solutions defined by the constraints.

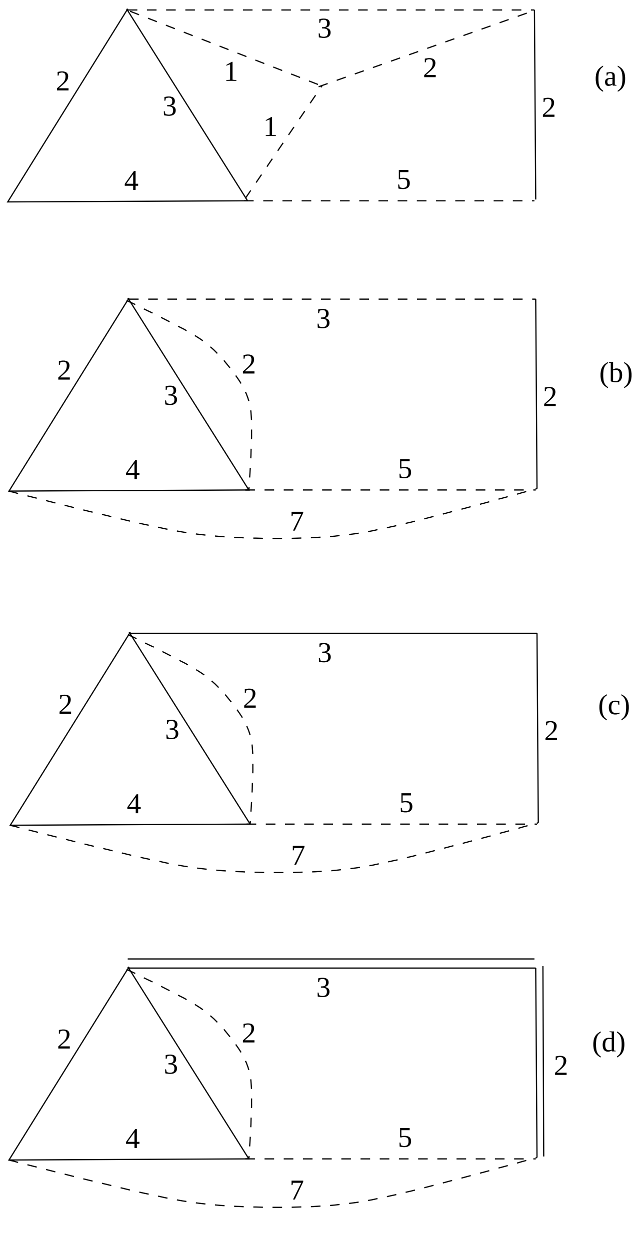


Figure 7.1: Algorithm for the undirected RPP

7.2 Directed RPP

Similar to the undirected RPP, given a weighted, directed, connected graph $G = (V, A)$ and a subset of arcs R with cost function c on the arcs, the goal is to find a minimum-cost directed tour that traverses each arc of R at least once. Lenstra and Rinnooy Kan [32] proved that directed RPP is NP-complete by providing a reduction from directed Hamiltonian circuit problem.

Similar to the algorithm for the undirected RPP, G is converted to $G' = (V', A')$ where V' is the set of nodes incident to R and A' contains R and the arcs that correspond to the shortest directed paths between every pair of nodes in V'. G' must be strongly connected for the problem to be feasible. Let R define the following connected components in G': $G_1, G_2 ... G_P$ with the respective vertex sets. Christofides *et al* [8] proposed the following heuristic for the directed RPP:

- Construct a shortest spanning arborescence (SSA) rooted at an arbitrary vertex, and connecting $G_1, ..., G_p$. Let H be the resulting graph.
- Run the minimum-cost flow algorithm and add arcs to H so that each node satisfies the symmetry condition ($indegree = outdegree$).
- Replace the shortest path arcs in the solution to the corresponding arcs of G. Determine a directed Eulerian tour in the output graph.

The above procedure can be repeated by considering each vertex as the root of the SSA. Similar to undirected RPP, the first step is not required if (V, R) is connected. In this case, run the minimum-cost flow problem directly to get an optimal solution. Christofides *et al* [8] have also provided IP formulations similar to that of the undirected RPP to solve the directed RPP optimally.

Even though Christofides presented a practically attractive heuristic algorithm for directed RPP, it does not have a bounded approximation ratio. Historically, it has been difficult to come up with algorithms with reasonable approximation ratios for problems with asymmetric edge costs compared to their symmetric counterparts. For example, Christofides designed an 1.5-approximation algorithm for undirected TSP when triangular inequality is satisfied [5]. However, for directed TSP, there is no approximation algorithm known with a proven approximation ratio better than $\log |V|$ [17].

We can get the same approximation ratio of $\log |V|$ for the directed RPP also. Given an instance of directed RPP, we can reduce it to the directed TSP: Let $G = (V, A)$ with $R \subset A$ be the instance of the directed RPP. Construct $H = (U, B)$ as follows:

- Let $R = \{(i_1, j_1), (i_2, j_2), \cdots, (i_n, j_n)\}$. For each arc (i_k, j_k) in R, introduce a node u_k in H.
- For every two nodes j_k and i_l, add an arc (u_k, u_l) in H whose cost is equal to the cost of a shortest path from j_k to i_l.
- Now solve the directed TSP on H, and map the solution back to directed RPP.

Note that this is an approximation preserving reduction from directed RPP to directed TSP. The instance of the directed TSP can be solved by an algorithm described by Fisher, Nemhauser, and Wolsey [17]. Since that algorithm is a $\log |V|$ approximation algorithm [17], we get the same performance bound for directed RPP.

7.3 Stacker crane problem

The stacker crane problem (SCP) is defined on a mixed graph $G = (V, E, A)$ where V is a set of nodes, E is a set of edges and A is a set of arcs. The problem is to determine a shortest tour including each arc at least once. The arcs of A can be considered as the movements to be performed by a crane, each exactly once, in a specified direction. The arcs can also be interpreted as deliveries to be made by a vehicle. The SCP then amounts to minimizing the total delivery costs. The TSP reduces to the SCP in which each node of the TSP is modeled by an arc of the SCP with cost 0. Hence the SCP is NP-hard.

Frederickson *et al* [21] proposed two heuristics and proved that the best solution from those heuristics is no worse than 9/5 times the cost of the optimal solution. We will provide the intuitive ideas behind the algorithm. Refer to the survey paper by Eiselt *et al* [13], and Frederickson, *et al* [21] for more details. The first heuristic LARGEARCS handles the case in which the arc costs are big and comparable to the cost of optimal solution: Find a minimum-cost flow to make the graph induced by the arcs symmetric, but the resulting output may contain disconnected components. Add minimum-cost edges to connect these components. The second heuristic SMALLARCS handles the case in which arc costs are small compared to the cost of an optimal solution. This heuristic contracts the arcs to nodes, and finds a TSP tour of the nodes. It then re-expands the arcs, and finds an Eulerian tour ignoring the direction of arcs. Consider both orientations of the tour, and add 2 copies of each arc that goes against the direction of the tour to make the resulting tours symmetric. SMALLARCS then outputs the oriented tour with the minimum cost. Now, choose the minimum cost solution between the solutions of LARGEARCS and SMALLARCS. Frederickson*et al* proved that the performance ratio of this algorithm is 9/5. Lukka and Salminen [33] presented another heuristic algorithm based on the heuristics for TSP. However, its worst-case performance ratio is unbounded.

CHAPTER 8

OTHER CPP VARIATIONS

In this chapter, we briefly review other variations of CPP not covered earlier.

8.1 Multiple postman problem

The multiple postman problem, also known as k-CPP, reflects more of a flavor of the real world problems than the 1-person problem. When one person cannot cover a vast domain, $k > 1$ persons are dispatched to collectively cover all the edges, under the constraint that all k persons share the load roughly equally. Similar problems can be defined for other problems too: k-traveling salesmen, k-stacker crane, etc. Even though the total traversal cost is still important, minimizing the maximum of the k-person tour costs takes precedence in many practical situations.

Given an undirected weighted connected graph $G = (V, E)$ with cost function c on edges, the k-CPP problem is to find a set of k sub-tours which collectively cover all the edges at least once. Frederickson *et al* [21] presented a reduction from the k-partition problem [46] to k-CPP, and proved that k-CPP is NP-complete. They also presented the following approximation algorithm for k-CPP: Find an optimal 1-CPP tour, and split the tour into k parts so that i-th postman can directly go to the starting node of the ith part of the tour using a shortest path, follow the tour path to the end node of the ith part, and return to the source node using a shortest path. They proved that the approximation ratio for their algorithm is $2 - 1/k$. Figure 8.1 shows an example for a 4-CPP. The optimal postman tour is split into 4 parts. The first postman starts the tour at v, follows the postman tour path to a, and returns to the starting node v via a shortest path from a. The second postman goes from v to a via a shortest path, follows the postman tour from a to b, and returns to v via a shortest path from b. Similarly, the third and fourth postmen cover b to c and c to d respectively.

8.2 Capacitated arc routing problem (CARP)

In this problem, each arc (v_i, v_j) has a nonnegative weight q_{ij}, and all arcs with positive weight must be traversed by a fleet of identical vehicles of capacity W. The rural postman problem is a special case of the CARP. Since most practical applications contain capacity restrictions, CARP is an important problem in the area of routing. The objective is to determine a minimum-cost traversal of all arcs by vehicles so that the total demand of all arcs serviced by any vehicle does not exceed W. Note that a vehicle may traverse an arc without servicing it, but each arc must be serviced by exactly one vehicle. Refer to the survey article by Eiselt *et al* [13] for details on a heuristic algorithms for, and IP formulations of, CARP.

8.3 General routing problem

Given an undirected graph $G = (V, A)$, $V' \subset V$, and $A' \subset A$, general routing problem (GRP) is to find a minimum cost tour which traverses all arcs of A'

and all nodes of V'. Clearly, TSP, CPP, and RPP are all special cases of GRP. Orloff [38] presented an IP formulation and a branch and bound algorithm to solve the GRP. The LP relaxation of the GRP is simply a matching problem and constraints are added for sub-tour elimination. Note that we can reduce the directed GRP to a directed TSP problem using a reduction similar to the reduction presented for the directed RPP in Section 7.2, and get a worst case bound of $\log |V|$.

8.4 Hierarchical postman problem

The hierarchical postman problem (HPP) is defined on a directed or on an undirected graph with a source node and a sink node. Let us consider the directed version with input graph $G = (V, A)$ with a source node s and a sink node t. The arcs of the graph are partitioned into k disjoint sets $A_1, A_2, A_3, ...A_k$, and an order relation $\prec$ is imposed on the elements of the partition. The HPP consists of determining a least-cost traversal of G starting at s, ending at t, and servicing arcs of the partition in such a way that if $A_i \prec A_j$, then all edges/arcs of A_i are serviced before any arc of A_j. However, arcs of A_j can be traversed before some arcs of A_i. This problem arises naturally in snow plowing operations where the subsets correspond to streets with different priority levels. Other applications are waste collection and flame cutting.

Consider the subgraphs $G_p = (V_p, A_p)$ induced by the edges of A_p. If each such subgraph is connected and the order relation is complete, i.e., a total order $A_1 \prec \ldots \prec A_k$ is specified, Dror, Stern and Trudeau [10] showed that the problem can be solved in polynomial time. Construct a directed acyclic graph (U, B) as follows: Define U_i as a cluster of nodes containing one copy of each

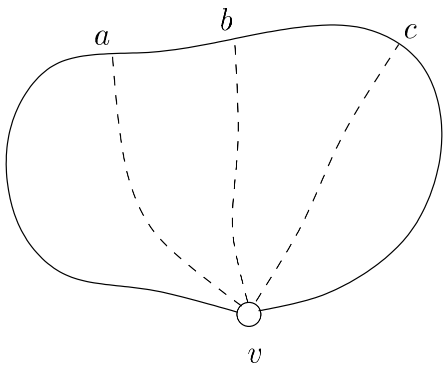


Figure 8.1: An example for 4-CPP

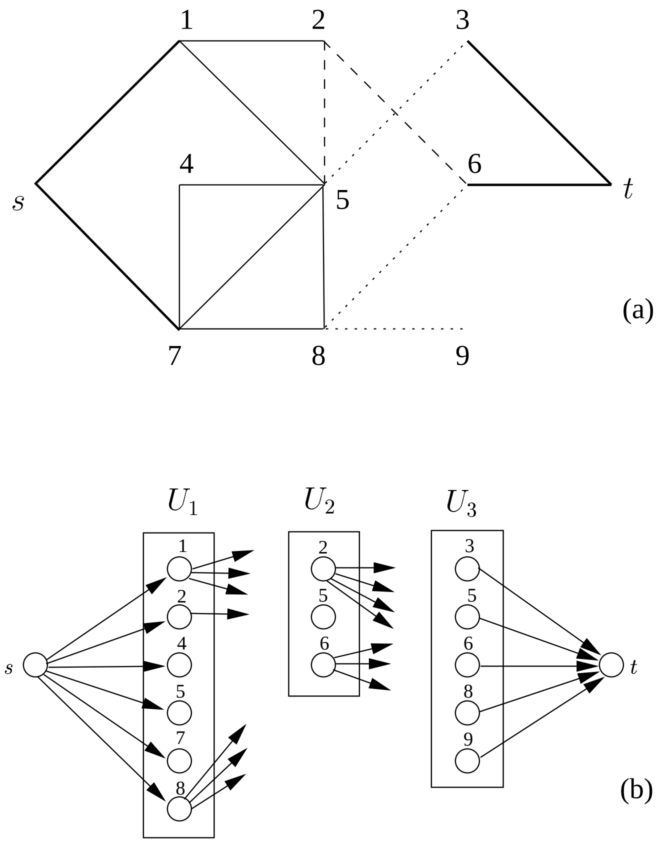


Figure 8.2: Hierarchical postman problem

vertex incident to an arc of A_i, and let $U = \{s,t\} \cup U_1 \cup U_2 \dots U_k$. The arc set B is made up of all arcs from s to the nodes of U_1 and from the nodes of U_k to t, and arcs from the nodes of U_i to the nodes of $U_{i+1} \forall i \in [1, k-1]$. The cost c_{uv} of an arc (u, v) in the new structure is defined as follows: If $u = s$ and $v \in U_1$, then c_{uv} is the shortest path from s to v. If $u \in U_i$ and $v \in U_{i+1}$ or $v = t$, then c_{uv} is the value of a minimum cost Eulerian path starting at u and ending at v covering all arcs of A_i. Now the goal is to compute a shortest path from s to t in (U, B) via the k clusters. Figure 8.2 (a) shows an example for the HPP. All arcs other than the ones from s and to t have associated priorities. The arcs in A_1 are in light solid lines, the arcs in A_2 are in dashed lines, and the arcs in A_3 are in dotted lines. Figure 8.2 (b) shows the corresponding nodes U_1, U_2, and U_3. All arcs are not shown explicitly.

Readers are referred to the articles by Dror *et al* [10], and Eiselt *et al* [13] for more complete descriptions of the algorithm. Dror *et al* [10] also proved that CPP on a graph with a general precedence relation is NP-complete. Intuitively, the precedence relations may specify only a partial order on the sets of arcs A_i's thereby allowing more flexibility in the order in which A_i's are traversed, making the optimization problem harder. Also, obtaining a lower bound on the cost of the optimal solution seems difficult.

8.5 Maximum benefit CPP

In this version of CPP, a benefit is derived from every traversal of a link. Additionally, the tour need not cover all the edges. Malandraki and Daskin [34] define a benefit function b_{ijk} associated with link (i, j) that gives the gross benefit of traversing the link for the k-th time. The benefit derived from traversing a link is expected to decrease as the number of traversals increase. Each link is also associated with a servicing cost c_{ij}^t and deadheading cost c_{ij}^d. The goal of the maximum benefit CPP is to minimize the total net cost of traversing the links or equivalently, maximize the the total net benefit.

Malandraki and Daskin [34] transformed G to an extended network G'. Figure 8.3 shows the transformation for one arc of the input graph. c_{ijk} specifies the traversal cost of the link (i, j) k-th time, and the capacity of the corresponding arc is 1 in G'. Note that $c_{ij1} < c_{ij2} < c_{ij3} < \cdots < c_{ijn} \leq c_{ij}^d$. One could try to solve the maximum benefit CPP by finding a minimum-cost flow algorithm in a suitably defined network, however the solutions may not be connected. Therefore, they formulate additional subtour-elimination constraints on an extended network and give a solution based on the branch and bound method.

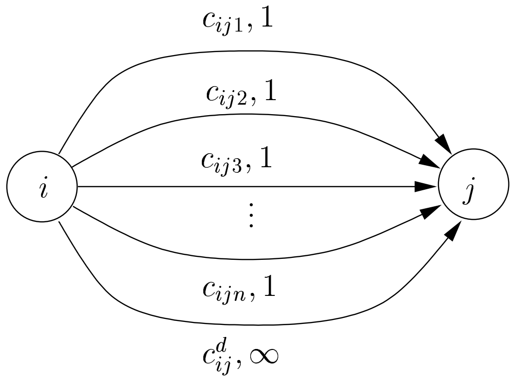


Figure 8.3: Maximum benefit CPP

CHAPTER 9

CONCLUSIONS

Problems of finding paths and tours on graphs have been widely studied due to their fundamental importance and practical applicability. In this dissertation:

- We presented a 3/2-approximation algorithm for the mixed postman problem. We analyzed the properties of intermediate solutions, improved the lower bound for the optimal solution, and modified Frederickson's mixed approach [20] to attain a new approximation algorithm.

- We presented a 3/2-approximation algorithm for the windy postman problem. We studied the properties of intermediate solutions, proved that our improved lower bound is valid even when the edge costs are asymmetric, and designed a new approximation algorithm.

- We presented a polynomial-time algorithm for the restricted mixed postman problem on even-degree graphs. We formulated it as a minimum-cost flow problem with capacity restrictions. Our algorithm either outputs an optimal solution, or indicates that the problem is infeasible for a given instance.

- We presented a 1.5-approximation algorithm for the restricted postman problem which can duplicate edges only. We used the minimum-cost flow approach on the edges to satisfy the imbalances created by the arcs, and finally added T-join edges to make all nodes have even degree.

- We proved that the directed Euler tour problem with restricted turns is NP-complete by presenting a simple reduction from the directed traveling salesman problem. We also observed that the mixed postman problem with duplication restricted to arcs is NP-complete based on the NP-hardness proof for the mixed postman problem presented by Papadimitriou [39].

- We observed that the worst case bound for the directed general routing problem and the directed Euler tour with restricted turns is $\log |V|$ by mapping these problems to the directed TSP using approximation preserving reduction procedures.

Our future work includes tuning our algorithms and applying them to real world data. There are still several open problems of practical importance:

- Find the necessary and sufficient conditions for a mixed graph to have a restricted MPP tour in which duplication is restricted to a specified subset of edges and arcs. Are there polynomial-time algorithms to detect feasibility? Are there approximation algorithms for the restricted MPP on general mixed graphs?

- Prove that the MPP with duplication restricted to edges is NP-hard. Are there approximation algorithms for this problem with approximation ratio better than 1.5?

- Find the necessary and sufficient conditions for a mixed graph to have a MPP tour with duplication restricted to arcs. Are there approximation algorithms for this problem?

- Prove or disprove that the following problems are NP-hard: undirected Euler tour with restricted turns, undirected Euler tour with weighted turns, undirected postman tour with restricted turns, and undirected postman tour with weighted turns. If they are NP-hard, are there approximation algorithms for these problems?

- Are there approximation algorithms for the stacker crane problem with an approximation ratio better than 9/5?

- Are there approximation algorithms for directed RPP whose worst case bounds are better than $\log |V|$?

BIBLIOGRAPHY

[1] Ahuja, R. K., Magnanti, T., and Orlin, J. *Network Flows.* Prentice Hall, 1993.

[2] Bland, R. G., Goldfarb, D., and Todd, M. J. The ellipsoid method: a survey. *Operations Research 29* (1981), 1039–1091.

[3] Bondy, J. A., and Murty, U. S. R. *Graph Theory with Applications.* North Holland, New York, 1980.

[4] Brucker, P. The Chinese postman problem for mixed networks. In *Proceedings of the International Workshop on Graph theoretic Concepts in Computer Science* (1980), vol. 100 of *LNCS*, Springer, pp. 354–366.

[5] Christofides, N. Worst case analaysis of a new heuristic for the traveling salesman problem. *Management Science Research Report No. 388* (1976).

[6] Christofides, N., Benavent, E., Campos, V., Corberan, A., and Mota, E. An optimal method for the mixed postman problem. In *System Modelling and Optimization* (1984), vol. 59 of *Notes in Control and Information Sciences*, Springer.

[7] Christofides, N., Campos, V., Corberan, A., and Mota, E. An algorithm for the rural postman problem. *Imperial College Report IC.O.R.81.5*

(1981).

[8] Christofides, N., Campos, V., Corberan, A., and Mota, E. An algorithm for the rural postman problem on a directed graph. *Mathematical Programming Study 26* (1986), 155–166.

[9] Corberan, A., and Sanchis, J. M. A polyhedral approach to the rural postman problem. *European Journal of Operational Research 79* (1994), 95–114.

[10] Dror, M., Stern, M., and Trudeau, P. Postman tour on a graph with precedence relation on arcs. *Networks 17* (1987), 283–294.

[11] Edmonds, J., and Johnson, E. L. Matching, Euler tours and the Chinese postman. *Mathematical Programming 5* (1973), 88–124.

[12] Eiselt, H. A., Gendreau, M., and Laporte, G. Arc routing problems, Part I: The Chinese postman problem. *Operations Research 43* (1995), 231–242.

[13] Eiselt, H. A., Gendreau, M., and Laporte, G. Arc routing problems, Part II: The rural postman problem. *Operations Research 43* (1995), 399–414.

[14] Euler, L. Solutio problematis ad geometrian situs pertinentis. *Commentarii Academiae Scientarum Petropolitianae 8* (1736), 128–140.

[15] Euler, L. Leonhard Euler and the Königsberg bridges. *Scientific American 189* (1953), 66–70.

[16] Even, S. *Graph Algorithms.* Computer Science Press, Potomac, Maryland, 1979.

[17] Fisher, M. L., Nemhauser, G. L., and Wolsey, L. A. An analysis of approximations for finding a maximum weight Hamiltonian circuit. *Operations Research 27* (1979), 799–809.

[18] Ford, L. R., and Fulkerson, D. R. *Flows in Networks.* Princeton U. Press, Priceton, N. J., 1962.

[19] Foulds, L. R. *Graph Theory Applications.* Springer-Verlag, New York, 1991.

[20] Frederickson, G. N. Approximation algorithms for some postman problems. *Journal of the Association for Computing Machinery 26* (1979), 538–554.

[21] Frederickson, G. N., Hecht, M. S., and Kim, C. E. Approximation algorithms for some routing problems. *SIAM Journal of Computing 7* (1978), 178–193.

[22] Frederickson, G. N., and JÁJÁ, J. On the relationship between the biconnectivity augmentation and traveling salesman problems. *Theoretical Computer Science 19* (1982), 189–201.

[23] Garey, M. R., and Johnson, D. S. *Computers and intractability: a guide to the theory of NP-completeness.* W. H. Freeman and Co., New York, 1979, p. 212.

[24] Grotschel, M., and Win, Z. A cutting plane algorithm for the windy postman problem. *Mathematical Programming 55* (1992), 339–358.

[25] Guan, M. Graphic programming using odd and even points. *Chinese Mathematics 1* (1962), 273–277.

[26] Guan, M. On the windy postman problem. *Discrete Applied Mathematics 9* (1984), 41–46.

[27] Guan, M., and Pulleyblank, W. Eulerian orientations and circulations. *SIAM Journal of Algorithms and Discrete Mathematics 6* (1985), 657–664.

[28] Karloff, H. *Linear Programming.* Birkhauser, Boston, 1991.

[29] Khachiyan, L. G. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady 20* (1979), 191–194.

[30] Lawler, E. L. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart, and Winston, 1976.

[31] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. *The Traveling Salesman Problem.* John Wiley, New York, 1985.

[32] Lenstra, J. K., and Rinnooy Kan, A. H. G. On general routing problems. *Networks 6* (1976), 273–280.

[33] Lukka, A., and Salminen, L. Comparison of three heuristics for an arc routing problem. *Research Report 6/1987* (1987).

[34] Malandraki, C., and Daskin, M. S. The maximum benefit Chinese postman problem and the maximum benefit traveling salesman problem. *European Journal of Operational Research 65* (1993), 218–234.

[35] Minieka, E. The Chinese postman problem for mixed networks. *Management Science 25* (1979), 643–648.

[36] Nemhauser, G. L., and Wosley, L. A. *Integer and Combinatorial Optimization*. John Wiley, 1988, pp. 648–651.

[37] Nobert, Y., and Picard, J. C. An optimal algorithm for the mixed Chinese postman problem. *Networks 27* (1996), 95–108.

[38] Orloff, C. S. A fundamental problem in vehicle routing. *Networks 4* (1974), 35–64.

[39] Papadimitriou, C. H. On the complexity of edge traversing. *Journal of the Association for Computing Machinery 23* (1976), 544–554.

[40] Pearn, W. L., and Li, M. L. Algorithms for the windy postman problem. *Computers & Operations Research 21* (1994), 641–651.

[41] Raghavachari, B., and Veerasamy, J. Approximation algorithms for mixed postman problem. In *Proceedings of 6th Integer Programming and Combinatorial Optimization* (1998), vol. 1412 of *LNCS*, Springer-Verlag, pp. 169–179.

[42] Raghavachari, B., and Veerasamy, J. Approximation algorithms for asymmetric postman problem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms* (1999), SIAM, pp. 734–741.

[43] Raghavachari, B., and Veerasamy, J. A 3/2-approximation algorithm for the mixed postman problem. *SIAM Journal of Discrete Mathematics* (to appear).

[44] Ralphs, T. K. On the mixed Chinese postman problem. *Operations Research Letters 14* (1993), 123–127.

[45] Roy, S., and Rousseau, J. M. The capacitated Canadian postman problem. *INFOR 27* (1989), 58–73.

[46] Sahni, S. K., and Gonzalez, T. P-complete approximation problems. *Journal of the Association for Computing Machinery 23* (1976), 555–565.

[47] Taha, H. A. *Operations Research.* MacMillan Publishing, New York, 1971.

[48] Tohyama, H., and Adachi, A. Complexity of a restricted Chinese postman problem. *Transactions of the Information Processing Society of Japan 37* (1996), 1886–1896.

[49] van Aardenne-Ehrenfest, T., and de Brujin, N. G. Circuits and trees in oriented linear graphs. *Simon Stevin 28* (1951), 203–217.

[50] Win, Z. *Contributions to Routing Problems.* Doctoral dissertation, Universitat Augsburg, Germany, 1987.

[51] Win, Z. On the windy postman problem on Eulerian graphs. *Mathematical Programming 44* (1989), 97–112.

[52] Yan, H., and Thompson, G. L. Finding postal carrier walk paths in mixed graphs. *Computational Optimization and Applications 9* (1998), 229–247.

VITA

Jeyakesavan Veerasamy, born in Puthuchennelkulam, Tamilnadu State, India on June 15, 1969, is the son of Mrs. Seeniammal and Mr. Veerasamy. He completed his schooling in St. Xaviers Higher Secondary School, Palayamkottai, in 1986, and then entered College of Engineering, Anna University, Madras, for higher studies. He received the degree of Bachelor of Engineering with a major in Electronics and Communications Engineering in May, 1990. He worked as a System Software Engineer at HCL Limited, Madras, from August 1990 to August 1991.

In August 1991, he joined the Graduate School of the University of Texas at Dallas and obtained the degree of Master of Science in Computer Science in December 1993. During that time, he worked on a research project, "Path based Restoration based Network Planning Tool," sponsored by Alcatel Network Systems, under the guidance of Dr. S. Venkatesan. He joined Northern Telecom in May 1994 as a member of scientific staff and worked on the Software for Wireless System Components based on North American Cellular Standards. Parallelly he pursued a Ph.D. in Computer Science at the University of Texas at Dallas. He left Northern Telecom in June 1997 and joined the CDMA Software Group in Samsung Telecommunications America. He married Priya Narayanasami in September 1998.

During the past five years he has 5 conference publications and 3 journal publications. Additionally, he has applied for 4 patents in the areas of Wireless Networking and Air-Interface Protocols.

This dissertation was typeset with LaTeX.