A PRIMAL-DUAL ALGORITHM FOR THE MAXIMUM CHARGE PROBLEM WITH CAPACITY CONSTRAINTS

SANGITA BHATTACHARJEE
Bachelor of Science, Shah Jalal University of Science and Technology, 2005

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

I dedicate this thesis to my **parents** and my **husband**.

Abstract

In this thesis, we study a variant of the maximum cardinality matching problem known as the maximum charge problem. Given a graph with arbitrary positive integer capacities assigned on every vertex and every edge, the goal is to maximize the assignment of positive feasible charges on the edges obeying the capacity constraints, so as to maximize the total sum of the charges. We use the primal-dual approach. We propose a combinatorial algorithm for solving the dual of the restricted primal and show that the primal-dual algorithm runs in a polynomial time.

Acknowledgments

I would like to express my profound gratitude to my supervisor Dr. Daya Gaur for his close supervision, inspiring guidance and helpful suggestions throughout the period of my Masters degree. I am indebted to him for all the support he provided me.

I would also like to express my deep acknowledgment to my co-supervisor Dr. Shahadat Hossain for his endless help and support in absence of my supervisor.

I would like to thank my M.Sc. supervisory committee member Dr. Hans-Joachim Wieden for his constructive suggestions. I would also like to thank my external examiner Dr. Ramesh Krishnamurti for his valuable suggestions and comments.

I am grateful to my supervisor and to the School of Graduate Studies for financial assistantships.

Finally I would like to thank my family members and my fellow graduate students Salimur Choudhury, Tarikul Alam Khan, Mahmudul Hasan and Sadid Hasan for their continuous encouragement and endless support that helped me to complete this thesis.

Contents

Approval/Signature Page ii

Dedication iii

Abstract iv

Acknowledgments v

Table of Contents vi

List of Tables viii

List of Figures ix

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Organization of the Thesis . . . 3

2 Linear Programming 4
- 2.1 Introduction . . . 4
- 2.2 Basic Concepts . . . 4
- 2.3 Simplex Method . . . 6
  - 2.3.1 Basic Feasible Solutions (bfs) . . . 7
  - 2.3.2 How to move from bfs to bfs . . . 8
  - 2.3.3 How to Obtain an initial bfs . . . 11
  - 2.3.4 Performance of the simplex algorithm . . . 13
- 2.4 Other methods for Linear Programming . . . 13
- 2.5 Duality . . . 13
- 2.6 Primal-Dual Method . . . 16
- 2.7 Primal-Dual Method and the Shortest Path Problem . . . 19

3 Related Research 23
- 3.1 Introduction . . . 23
- 3.2 Maximum Cardinality Matching (MCMP) . . . 23
  - 3.2.1 Cardinality Matching In Bipartite Graphs . . . 25
  - 3.2.2 Cardinality Matching in General Graphs . . . 26
  - 3.2.3 Integer Program for MCMP . . . 28
- 3.3 Maximum Fractional Matching . . . 28
- 3.4 Unconstrained Fractional Matching Problem (UFMP) . . . 29
  - 3.4.1 Problem formulation . . . 30
  - 3.4.2 Characterization of the optimal solution to the DRP_{ufm} . . . 32

3.4.3 Algorithm for solving DRP_{ufm} . . . 33
3.4.4 Primal-Dual Algorithm for UFMP . . . 38

4 Maximum Charge Problem **40**
4.1 Introduction . . . 40
4.2 Definitions . . . 40
4.3 Problem Formulation . . . 41
4.4 Primal-Dual interpretation . . . 42
4.5 Primal-Dual Algorithm . . . 46
4.5.1 Characterization of the Optimal Solution to the DRP_{mcp} . . . 46
4.5.2 Phased Approach for Solving the DRP_{mcp} . . . 49
4.5.3 Construction of the Hungarian Tree (HT) . . . 50
4.5.4 Topological Erase Approach for Finding the Augmenting Paths . . . 53
4.5.5 Blocking Charge Approach for Finding the Augmenting Paths . . . 55
4.5.6 Analysis of the Phased Approach . . . 64
4.6 Why choose the shortest augmenting paths . . . 66
4.7 Analysis of the Primal-Dual Algorithm . . . 70

5 Experiments and Results **74**
5.1 Introduction . . . 74
5.2 Data Sets . . . 74
5.3 Experimental Results . . . 75

6 Conclusions **86**
6.1 Summary . . . 86
6.2 Future Research . . . 87

A **88**
A.1 Implementation . . . 88
A.1.1 Implementation of the Topological Erase Approach . . . 92
A.1.2 Implementation of the Blocking Charge Approach . . . 93

Bibliography **95**

List of Tables

5.1 Experiments on random bipartite dense graphs 76
5.2 Experiments on random bipartite sparse graphs 77
5.3 Experiments on random general dense graphs 78
5.4 Experiments on random general sparse graphs 79

List of Figures

3.1 Matching Example 24
3.2 Example of a blossom 27
3.3 Example of a shrunken blossom 27
3.4 A general graph and its bipartite equivalent 34

4.1 A bipartite graph instance and the associated HT 51
4.2 An unsaturated edge shared by multiple augmenting paths 53
4.3 Blocking charge approach 56
4.4 An edge shared by two augmenting paths 65
4.5 A bipartite graph 67
4.6 Two optimal solution to the DRP_{mcp} 68
4.7 New charge solution $q+\varepsilon\bar{q}$ 68
4.8 Optimal solution to the current DRP_{mcp} instance 69
4.9 New charge solution $q+2\varepsilon\bar{q}$ 69

5.1 Comparison of the running time of the two different approaches on random bipartite dense graphs 77
5.2 Comparison of the running time of the primal-dual algorithm with two different approaches on random bipartite dense graphs 78
5.3 Comparison of the running time of the two different approaches on random bipartite sparse graphs 79
5.4 Comparison of the running time of the primal-dual algorithm with two different approaches on random bipartite sparse graphs 80
5.5 Comparison of the running time of the two different approaches on random general dense graphs 81
5.6 Comparison of the running time of the primal-dual algorithm with two different approaches on random general dense graphs 82
5.7 Comparison of the running time of the two different approaches on random general sparse graphs 83
5.8 Comparison of the running time of the primal-dual algorithm with two different approaches on random general sparse graphs 84

A.1 A graph, corresponding weighted adjacency list and the list of node capacities 89
A.2 A bipartite graph, corresponding weighted adjacency list and the list of node capacity 90

Chapter 1

Introduction

1.1 Introduction

In graph theory, a graph $G = (V, E)$ is a mathematical structure, consisting of a set of vertices V and a set of edges E. The set E represents the pairwise relationship among the vertices in V. For each edge $e = (u, v) \in E$ the two endpoints $u, v \in V$ are known as the neighbors of each other.

A *matching* in a graph is a set of edges such that no node is paired more than once. In the field of graph theory and combinatorial optimization *matching* is a topic of central interest for the researchers throughout the years.

The most basic problem in matching theory is the problem of finding a matching containing maximum number of edges and is known as the *maximum cardinality matching problem* (MCMP). There can be a number of variations of MCMP. Some of the variations arise when the edges of the graph instance are assigned arbitrary capacities [18] or the vertices of the graph instance are assigned arbitrary capacities [6] or the edges are allowed to take fractional values [5] or both the vertices and the edges are assigned arbitrary capacities. For any instance of MCMP both the edges and the vertices are considered to be assigned a capacity of 1. The MCMP and its variants have many applications, specially in communication and scheduling [19]. The algorithms for solving the matching problems use the concept of augmentation and for this reason these algorithms are closely related to the solutions for the *Maximum Network Flow* problems [1].

This thesis is about a generalization of the matching problem known as the *maximum charge problem*. For a *maximum charge problem* graph instance both the edges and the vertices are assigned positive arbitrary capacities. In 2006 Krishnamurti *et al.* [17] proved the Berge's theorem [3] for the *maximum charge problem* with capacity constraints. However the proof is nonconstructive. Their approach can be used to characterize the optimal solution to the *dual of the restricted primal* (DRP) in the *primal-dual framework* and is the objective of study in this thesis.

The *maximum charge problem* with capacity constraints can be represented using a linear programming formulation (See Ch. 4). Linear programming problems can be solved in polynomial time. A practical algorithm for solving linear programming problem is the *simplex* method. But in the worst case the *simplex* method may not terminate in polynomial time. Also the solution computed by *simplex* can be affected by numerical instability. To avoid numerical instability and to ensure strongly polynomial time termination we focus on designing a combinatorial algorithm based on the *primal-dual approach* for this problem. For many problems in graph theory with a linear programming formulation combinatorial algorithms have shown better performance as compared to the *simplex* algorithm.

The *primal-dual approach* provides a structure for designing efficient combinatorial algorithms for the *network flow* and the *matching* problems. In this thesis we use this method for designing a combinatorial algorithm for the *maximum charge problem*. We consider the *maximum charge problem* as the *dual* problem and formulate the *primal* problem. According to the *primal-dual approach* we formulate the *restricted primal* problem and the *dual of the restricted primal* (DRP) problem. We characterize the optimal solution to the DRP and propose a combinatorial approach for solving the DRP optimally. We propose two algorithms for solving the DRP optimally. We compare the algorithms both theoretically and experimentally. Finally we give the primal-dual algorithm that solves the *maximum charge*

problem by iteratively solving the associated DRP optimally. We also prove the polynomial running time complexity of the given algorithm.

1.2 Organization of the Thesis

In Chapter 2 we discuss the concept of *linear programming* and the *simplex* method for solving linear programming problems. We also discuss the concept of duality and the *primal-dual method* for solving linear programming problems.

In Chapter 3 we describe *matching* in graphs and related problems like the *maximum cardinality matching problem*, *fractional matching problem* and *unconstrained fractional matching problem*. We also discuss some of the related research that has been done on these problems.

In Chapter 4 we present the *maximum charge problem*. We introduce the primal-dual interpretation for the problem and discuss two combinatorial approaches for solving the DRP for the problem. We discuss the importance of our proposed approach. Last of all we analyze the proposed primal-dual algorithm and prove a polynomial running time bound.

In Chapter 5 we present the implementation of our primal-dual algorithm using both the combinatorial approaches for solving the DRP for the *maximum charge problem*. We compare both the approaches experimentally on dense and sparse random graphs.

Finally, we conclude the thesis in Chapter 6 with future research direction.

Chapter 2

Linear Programming

2.1 Introduction

In this chapter we discuss the concept of *linear programming* and some methods for solving the *linear programming problems*. Specifically we give our attention to the *primal-dual method* for solving the *linear programming problems*. For a detailed discussion please see the books by Chávtal [7], Vanderbei [25], Papadimitriou and Steiglitz [22].

In Section 2.2 we discuss the basic concepts of *linear programming*. In Section 2.3 and Section 2.4 we discuss the *simplex* method and some other methods for solving *linear programming problems* respectively. In Section 2.5 we discuss the concept of *duality*. In Section 2.6 we present the *primal-dual method* for the *linear programming problems* and in Section 2.7 we present an application of the *primal-dual method* on the *shortest path problem*.

2.2 Basic Concepts

A *linear programming problem* (LP) is the problem of minimizing or maximizing a linear objective function obeying a finite number of linear equalities and inequalities known as

the constraints. Typically a *linear programming problem* has the form:

$$\begin{aligned} &\text{minimize} && cx \\ &\text{subject to} && Ax \geq b \\ & && x \geq 0 \end{aligned} \tag{2.1}$$

This problem is an example of a minimization problem. A maximization LP problem has the form:

$$\begin{aligned} &\text{maximize} && cx \\ &\text{subject to} && Ax \leq b \\ & && x \geq 0 \end{aligned} \tag{2.2}$$

The linear function cx is the objective function where c and x represent the cost vector and the vector of decision variables respectively. A represents an $m \times n$ integer matrix known as the coefficient matrix and b represents the right hand side vector, a_{ij} represents the element of the ith row and jth column of A. A solution is an assignment of values to the elements of x. A solution that satisfies all the constraints is called a feasible solution. A feasible solution with a minimum objective function value is called the optimal solution to the minimization problem and a feasible solution with a maximum objective function value is called the optimal solution to the maximization problem. If a solution does not satisfy the constraints then the solution is called infeasible. A problem is infeasible if it has no feasible solution. An LP in the form of equations 2.1 and 2.2 are said to be in *canonical*

form.

$$\begin{aligned} &\text{minimize} && cx \\ &\text{subject to} && Ax = b \\ & && x \geq 0 \end{aligned} \tag{2.3}$$

$$\begin{aligned} &\text{maximize} && cx \\ &\text{subject to} && Ax = b \\ & && x \geq 0 \end{aligned} \tag{2.4}$$

The LP in the form of equation 2.3 and 2.4 are said to be in *standard form*. Actually the *standard* and *canonical forms* are equivalent. An instance of LP in *canonical form* can be transformed to *standard form* using a series of slack and surplus variables [22].

2.3 Simplex Method

The *simplex* method is an iterative method for solving the *linear programming problems*. The method was devised by George B. Dantzig [8]. The operation of the *simplex* method requires an LP to be in *standard form*. So if the original problem is in *canonical form* it is transformed to the *standard form* by adding (or subtracting) the slack variables to the inequality constraints. Each slack variable also appears in the objective function with coefficient $c_s = 0$.

Given an LP problem in *standard form* the *simplex* method starts by finding an initial basic feasible solution and moves from one basic feasible solution to another by using a series

of row operations until an optimal solution is obtained. The step of moving between basic feasible solutions is known as *pivoting*.

In this section we will discuss the simplex method considering the LP for a minimization problem.

2.3.1 Basic Feasible Solutions (bfs)

Before discussing the simplex method in detail we need to know what is a basic feasible solution (bfs). We start with the assumption that the $m \times n$ $(m < n)$ constraint matrix A always contains m linearly independent columns A_j which forms a $m \times m$ nonsingular matrix $B = [A_{j_k}]$ for $k = 1, 2, \ldots, m$ known as the basis $\mathcal{B}$ of A. The decision variables corresponding to the basis are called *basic variables* and the others are called *nonbasic variables*. A bfs corresponding to a basis $\mathcal{B}$ is computed by setting all the nonbasic variables to zero and then by solving m resulting equations to determine the value of the basic variables. After setting all nonbasic variables to zero the constraint equation $Ax = b$ in the LP 2.3 or 2.4 takes the form:

$$Bx_{\mathcal{B}(i)} = b \tag{2.5}$$

where $\mathcal{B}(i)$ represents the index of ith basic variable. So we can write:

$$x_{\mathcal{B}(i)} = B^{-1}b \tag{2.6}$$

And thus we have the values of the basic variables in the bfs corresponding to B.

For example consider the following LP problem in standard form:

$$
\begin{aligned}
\text{minimize} \quad & 2x_1 + 3x_2 + 4x_3 + 2x_4 + x_6 \\
\text{subject to} \quad & 2x_2 + 3x_3 + x_4 = 5 \\
& x_1 + x_2 + 2x_3 + x_5 = 4 \\
& x_1 + 2x_2 + 3x_3 + x_6 = 7 \\
& x_1, x_2, x_3, x_4, x_5, x_6 \geq 0
\end{aligned}
$$

Here a basis can be formed using the columns A_4, A_5, A_6 and $B = I$[1], the corresponding basic variables are x_4, x_5 and x_6. Setting all nonbasic variables to zero the bfs corresponding to $\mathcal{B}$ is $x = \{0,0,0,5,4,7\}$.

2.3.2 How to move from bfs to bfs

Starting with an initial bfs the *simplex* algorithm progresses by moving to another bfs with smaller objective function value in search of an optimal solution. From Section 2.3.1 we know that any LP instance with coefficient matrix A contains m columns corresponding to the basis $\mathcal{B}$, that is the vector x of decision variables contains m basic variables and $(n-m)$ nonbasic variables. We denote the set of basic variables as x_B and the set of nonbasic variables as x_N. During each transition from one bfs to another the *simplex* method ensures exactly one column of A corresponding to some basic variable comes out of the basis and one column corresponding to a nonbasic variable enters the basis. Now we will discuss the rules of choosing the appropriate column of A to enter the basis and appropriate column to leave the basis.

[1] I represents the identity matrix

Given an initial bfs x and the basis $\mathcal{B}$, suppose we want to bring a column $A_j \notin \mathcal{B}$ into the basis. Then the nonbasic variable x_j corresponding to A_j is called the entering variable. This process is carried out in order to improve the value of the objective function. So it is necessary to choose a column which is the most "profitable" one. If x_j is chosen as the entering variable then its value can be increased from zero to some positive value. To maintain feasibility we need to ensure that each unit increase in the value of x_j causes the same amount of each basic variable to decrease. Thus each unit change in x_j causes a net change in the cost of the column j given by:

$$\bar{c_j} = c_j - \sum_{i=1}^{m} a_{ij} c_{\mathcal{B}(i)} = c_j - z_j \tag{2.7}$$

c_j is called the relative cost of column j. A column j is "profitable" to bring in the basis if $\bar{c_j} < 0$. If $\bar{c_j} \geq 0 \ \ \forall j \in x_N$ then the resulting solution is optimal.

Once a column $A_j \notin \mathcal{B}$ enters the basis we need to remove a column $A_{\mathcal{B}(i)}$ from the basis. The variable $x_{\mathcal{B}(i)}$ corresponding to the column leaving the basis is called the leaving variable. This process is carried out in order to obtain the largest decrease in the objective function value while ensuring that the value of the current basic variables remain nonnegative.

When the value of x_j increases from zero to some positive value the basic variables take the value:

$$x_{B(i)} = b_i - a_{ij} x_j \qquad \text{for } i \in \mathcal{B} \tag{2.8}$$

In order to maintain feasibility we ensure that the value of x_j can be raised up to:

$$x_j = \min_{a_{ij}>0} \frac{b_i}{a_{ij}} \qquad \text{for } i \in \mathcal{B} \tag{2.9}$$

This x_j value causes at least one basic variable to become zero and we choose that specific $x_{\mathcal{B}(i)}$ as the leaving variable and the corresponding column $A_{\mathcal{B}(i)}$ as the leaving column.

We can summarize the rules for choosing the column entering the basis and the column leaving the basis as follows:

- Choose a column $A_j \notin \mathcal{B}$ to enter the basis if the relative cost for that column $\bar{c_j} < 0$.
- Choose a column $A_{\mathcal{B}(i)}$ to leave the basis if for the specific row i, $a_{ij} > 0$ and $\frac{b_i}{a_{ij}}$ is minimum over all $i \in \mathcal{B}$. Here j is the index of the entering column and i is the index of the row corresponding to the leaving basic column.

Once the entering and the leaving columns and their corresponding entering and leaving variables have been selected we can move from one bfs to another using appropriate row operations.

Now we discuss two special conditions that might occur during the pivot operation:

- **Case 1**: If all $a_{ij} \leq 0$, $i \in \{1, 2, \ldots, m\}$ then the value of the entering variable x_j can be increased indefinitely without violating feasibility and an arbitrarily small objective function value can be produced. In this case the LP problem is called unbounded.
- **Case 2**: A basic feasible solution is called degenerate if it contains more than $n - m$ zeros that means some of the basic variables have the zero value. If the current basic feasible solution is degenerate then after carrying out the pivot operation we come over with a basic feasible solution with same value although the basis is different.

The problem of degeneracy may sometimes cause the *simplex* method to arrive at a previously visited basis after a few pivot operations. This incident causes the simplex method to cycle in an infinite loop. We can avoid the problem of cycling by following a proper pivoting rule. One such pivoting rule is due to R. G. Bland [4]. This rule is known as

Bland's anti-cycling rule. It stipulates the following criteria for selection of entering and leaving columns:

- Choose a column indexed j to enter the basis if $\bar{c_j} < 0$ and j is the smallest index.
- Choose the smallest indexed columns $\mathcal{B}(i)$ to leave the basis if for the corresponding row i, $a_{ij} > 0$ and $\frac{b_i}{a_{ij}}$ is minimum over all $i \in \{1, 2, \ldots, m\}$.

The algorithm 2.1 provides a brief outline of the *simplex* algorithm using *Bland's rule* for pivoting.

Algorithm 2.1 The Simplex Algorithm

Require: An LP in standard form
Ensure: Optimal solution to the LP

```
Set Optimal = No
while Optimal = No and Unbounded = No do
    if c̄_j ≥ 0 for all j then
        Optimal = Yes
    else
        Choose the column indexed by the smallest j with c̄_j < 0
        if a_ij ≤ 0 for all i ∈ {1,2,...,m} then
            Unbounded = Yes
        else
            Find out the row i a_ij > 0 and the ratio b_i/a_ij is minimum. If there is a tie than
            choose i with smaller index
            Pivot on a_ij
        end if
    end if
end while
```

2.3.3 How to Obtain an initial bfs

The *simplex* method requires an initial bfs to start with. Sometimes an initial bfs can be inherited as a part of the problem formulation. For an LP formulated in canonical form after converting it to the standard form the slack variables constitutes the initial basis. If

this is not the case then the *two phase method* is used. In this method an auxiliary problem is introduced as follows:

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{m} t_i && \\ &\text{subject to} && \sum_{j=1}^{n} a_{ij}x_j + t_i = b_i && i = 1,2,\ldots,m \\ & && x_j \geq 0 && j = 1,2,\ldots,n \\ & && t_i \geq 0 && i = 1,2,\ldots,m \end{aligned} \tag{2.10}$$

Here $t_i,\ i = 1,2,\ldots,m$ is the new artificial variable introduced for each constraint.

- In phase I we solve the auxiliary problem using the ordinary *simplex* algorithm. After solving the auxiliary problem any of the following three cases can occur depending on the objective function value ζ and the existence of the artificial variables in the basis.

    1. If $\zeta = 0$ and all artificial variables are driven out of the basis then we have a basic feasible solution to the original problem and we can continue with the phase II.
    2. if $\zeta > 0$ then the original problem has no feasible solution.
    3. if $\zeta = 0$ but some artificial variable remains in the basis then we choose the column of the basis corresponding to an artificial variable. As $\zeta = 0$ so the row corresponding to the artificial variable column has $b = 0$. Hence we can pivot on any nonzero element along the row corresponding to a non-artificial variable and this pivot operation does not cause any change in ζ. In this way we drive the artificial variable out of the basis. We repeat this process until a feasible basis with all original variables is obtained. Then we continue in phase II. This

approach of driving out the artificial variable out of basis sometimes can result a row with zero elements in all columns. In such situation we drop the row out and continue in phase II with a basis of lower dimension.

- In phase II we reintroduce original objective function, drop off all the artificial variables and start the simplex method to solve the original problem with the basis resulting from phase I as the initial basis.

2.3.4 Performance of the simplex algorithm

Although the *simplex* method runs in polynomial time and so works well in practice but there are examples for which this method can require exponential time [15].

2.4 Other methods for Linear Programming

The first polynomial time algorithm for the *linear programming problems* was given by Khachiyan [14] in 1979 by adapting the *ellipsoid method*. Karmarkar [12] gave a polynomial time algorithm for LP based on interior point method.

2.5 Duality

The basic concept of *duality* is that every LP problem has another problem associated with it. The original problem is called the primal and the associated problem is called the dual.

Given a minimization LP problem in *canonical form*,

$$\begin{aligned} &\text{minimize} && cx \\ &\text{subject to} && Ax \geq b \\ & && x \geq 0 \end{aligned} \tag{2.11}$$

The associated dual is,

$$\begin{aligned} &\text{maximize} && \pi b \\ &\text{subject to} && \pi A \leq c \\ & && \pi \geq 0 \end{aligned} \tag{2.12}$$

The dual problem provides a bound for the primal objective function. For instance the objective function in (2.12) provides a lower bound for the objective function in (2.11). This fact is true in general and known as the *weak duality theorem*.

Theorem 2.5.1. *(Weak duality theorem) If $x = [x_1, x_2, \ldots, x_n]$ is a feasible solution to the primal in (2.11) and $\pi = [\pi_1, \pi_2, \ldots, \pi_m]$ is a feasible solution to the dual in (2.12) then,*

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} \pi_i b_i$$

Proof. We have

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} \pi_i a_{ij} \right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) \pi_i \geq \sum_{i=1}^{m} \pi_i b_i$$

Hence the theorem. □

The *strong duality theorem* states that at optimality the objective function values of the pri-

mal and the dual are equal. For a proof of the *strong duality theorem* please see Chávtal [7] or Vanderbei [25].

Theorem 2.5.2. *(Strong duality theorem) If there exists an optimal solution* $x = [x_1, x_2, \ldots, x_n]$ *to the primal problem then there must exist an optimal solution* $\pi = [\pi_1, \pi_2, \ldots, \pi_m]$ *to the corresponding dual problem such that,*

$$\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} \pi_i b_i$$

The *strong duality theorem* states that if the primal has an optimal solution then the dual also has one. If the primal solution is unbounded then according to the *weak duality theorem* the dual solution is infeasible. Similarly, if the primal solution is infeasible than the dual solution is unbounded. So we can say that the *strong duality* and the *weak duality* theorems state the fact that there always exists a balance between the primal and the dual solutions.

Given feasible solutions x and π to the primal and the dual respectively the necessary and sufficient conditions for simultaneous optimality of both x and π are known as the *complementary slackness conditions*.

Theorem 2.5.3. *(Complementary slackness) A feasible primal-dual pair* x *and* π *is optimal if and only if*

$$\pi_i(a_i x - b_i) = 0 \quad i = 1, 2, \ldots, m \tag{2.13}$$

$$(\pi A_j - c_j)x_j = 0 \quad j = 1, 2, \ldots, n \tag{2.14}$$

Here a_i denotes the ith row and A_j denotes the jth column of the coefficient matrix A. The *complementary slackness conditions* play a vital role in the design of algorithms using the *primal-dual method*.

2.6 Primal-Dual Method

The *primal-dual method* is a general method for solving the *linear programming problems*. In 1955 Kuhn [18] devised this method in an attempt to solve the *assignment problem*.

Starting with a dual feasible solution π the *primal-dual method* searches for a feasible solution x in primal satisfying the *complementary slackness condition* with respect to π. If such a solution is found, then x and π forms a pair of optimal solutions. Otherwise the method prescribes a modification of π and iterates.

In this section we give a detailed description of the *primal-dual algorithm* based on the representation by Papadimitriou and Steiglitz [22].

Let us consider the primal problem (P) in standard form:

$$\begin{aligned} &\text{minimize} && cx && \text{(P)} \\ &\text{subject to} && Ax = b \\ & && x \geq 0 \end{aligned}$$

where $b \geq 0$ and this can be ensured by multiplying the equalities where $b < 0$ in P by -1.

The corresponding dual problem (D) can be constructed as follows:

$$\begin{aligned} &\text{maximize} && \pi b && \text{(D)} \\ &\text{subject to} && \pi A \leq c \\ & && \pi \leqq 0 \end{aligned}$$

The dual decision variables π are unconstrained due to the equality constraint in P.

Let us revisit the *complementary slackness conditions* from the previous section:

$$\pi_i(a_i x - b_i) = 0 \quad i = 1,2,\ldots,m \tag{2.15}$$

$$(\pi A_j - c_j)x_j = 0 \quad j = 1,2,\ldots,n \tag{2.16}$$

Since P is in standard form, any feasible solution x in P will satisfy the set of conditions in (2.15). So our interest is in satisfying the set of conditions in (2.16).

Given a dual feasible solution π our main goal is to work towards feasibility in primal. For a dual feasible solution π we define the set J of *tight* constraints in the dual D as follows:

$$J = \{j \mid \pi A_j = c_j\} \tag{2.17}$$

According to the equation (2.16) for the set of dual constraints $j \in J$ the corresponding primal variable x_j can be set to any arbitrary value satisfying the equation (2.15). But for the set of dual constraints $j \notin J$ we have to set the corresponding primal variable $x_j = 0$. In order to satisfy the dual constraints $j \in J$ and to gain feasibility in primal we define a new LP called the restricted primal(RP) as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{i=1}^{m} w_i && && \text{(RP)}\\
&\text{subject to} && \sum_{j\in J} a_{ij}x_j + w_i = b_i && i = 1,2,\ldots,m\\
& && x_j \geq 0 && \forall j \in J\\
& && x_j = 0 && \forall j \notin J\\
& && w_i \geq 0 && i = 1,2,\ldots,m
\end{aligned}$$

Here we add a slack variable w_i to each constraint in P in order to gain feasibility in the primal. The RP can be solved using the ordinary simplex method. If the optimal solution to RP is zero then $w_i = 0$ for all primal constraints and hence we have found a feasible and optimal solution to the primal. If the optimal solution to RP is positive then we have to consider the dual of the restricted primal (DRP):

$$\begin{array}{lll} \text{maximize} & \pi b & \text{(DRP)} \\ \text{subject to} & \pi A_j \leq 0 \qquad & \forall j \in J \\ & \pi_i \leq 1 \qquad & i = 1,2,\ldots,m \\ & \pi_i \lesseqgtr 0 & \end{array}$$

Typically DRP is the simplified version of the dual D and can be solved by simpler combinatorial algorithm. We denote an optimal solution to the DRP by $\bar{\pi}$. If the DRP optimal solution has nonzero value then it suggests an improvement to the present dual solution π by a linear combination of π and $\bar{\pi}$:

$$\pi_{new} = \pi + \theta\bar{\pi} \tag{2.18}$$

Here π_{new} is the improved solution to the dual. Choosing an appropriate θ is very important in the calculation of π_{new} in order to maintain feasibility in D and to ensure an improvement in the objective function value.

If we multiply both side of the equation (2.18) with b then we have:

$$\pi_{new} b = \pi b + \theta\bar{\pi} b \tag{2.19}$$

As RP-DRP is a primal-dual pair and the optimal solution to RP is positive so $\bar{\pi}b > 0$. We choose $\theta > 0$ to improve the objective function value in D. Now the question is about the upper bound on θ.

By multiplying both side of the equation (2.18) with A_j

$$\pi_{new}A_j = \pi A_j + \theta\bar{\pi}A_j \tag{2.20}$$

From the DRP we have $\bar{\pi}A_j \leq 0$ for all $j \in J$. So for the tight constraints in D we can increase θ indefinitely without violating the feasibility in D because:

$$\pi_{new}A_j = \pi A_j + \theta\bar{\pi}A_j \leq c_j \tag{2.21}$$

If $\bar{\pi}A_j \leq 0$ for all $j \notin J$ then also we cannot find an upper bound on θ. In fact this case implies that D is unbounded and hence P is infeasible. Finally if there exists some dual constraints $j \notin J$ for which $\bar{\pi}A_j > 0$ then we can find a value of θ maintaining the feasibility in D as follows:

$$\theta = \min_{\substack{j \notin J \\ such\ that \\ \bar{\pi}A_j > 0}} \left[\frac{c_j - \pi A_j}{\bar{\pi}A_j} \right] \tag{2.22}$$

This θ value will ensure at least one dual constraint becomes "tight" and enters the set J at the beginning of the next iteration. The whole process described above is repeated until a RP-DRP pair of optimal objective function value 0 is obtained.

2.7 Primal-Dual Method and the Shortest Path Problem

In this section we illustrate the *primal-dual method* by applying it to the *shortest path problem*. We use the presentation due to Papadimitriou and Steiglitz [22]. Given a weighted

directed graph $G = (V, E)$ the shortest path problem is to find a path between two distiguished vertices $s \in V$ and $t \in V$ such that the total weight along the path is minimized. The shortest path problem can be formulated as the following primal problem:

$$\begin{aligned} &\text{minimize} && cx && (P_{\text{spath}}) \\ &\text{subject to} && Ax = \begin{bmatrix} +1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} \\ & && x \geq 0 \end{aligned}$$

Here A is the $m \times n$ vertex-edge incidence matrix $[a_{ij}]$ of G. m is the total number of vertices and n is the total number of edges in G. $a_{ij} = +1$ if the edge $e_j \in E$ leaves the node $v \in V$, $a_{ij} = -1$ if the edge $e_j \in E$ enters the node $v \in V$, $a_{ij} = 0$ otherwise. $x \in \Re^n$ is the vector of the edge variables and $c \in \Re^n$ is the vector of the weights associated with each edge $e \in E$. The dual of the shortest path problem is given by:

$$\begin{aligned} &\text{maximize} && \pi_s - \pi_t && && (D_{\text{spath}}) \\ &\text{subject to} && \pi_u - \pi_v \leq c_e && \forall e = (u, v) \in E \\ & && \pi_v \lesseqgtr 0 && \forall v \in V \end{aligned}$$

Given a feasible solution π to the D_{spath} the set J of tight constraints in D_{spath} can be defined as:

$$J = \{e \mid \pi_u - \pi_v = c_e\} \tag{2.23}$$

Given the set J the restricted primal can be formulated as :

$$\begin{aligned}
&\text{minimize} && \sum_{v \in V} w_v && && (RP_{\text{spath}}) \\
&\text{subject to} && A_J x_J + w_v = \begin{bmatrix} +1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} && && \\
& && x_e \geq 0 && \forall e \in J \\
& && x_e = 0 && \forall e \notin J \\
& && w_v \geq 0 && \forall v \in V
\end{aligned}$$

Here A_J is the matrix containing only the columns from A which are the members of set J and x_J is the vector containing only the variables corresponding to set J. The Dual of the restricted primal is given by:

$$\begin{aligned}
&\text{maximize} && \pi_s - \pi_t && && (DRP_{\text{spath}}) \\
&\text{subject to} && \pi_u - \pi_v \leq 0 && \forall e = (u,v) \in J \\
& && \pi_v \leq 1 && \forall v \in V \\
& && \pi_v \lesseqgtr 0 && \forall v \in V
\end{aligned}$$

According to the formulation we can characterize the optimal solution to the DRP_{spath} as follows:

$$\bar{\pi} = \begin{cases} 1 & \text{for all vertices reachable from } s \text{ by the edges in } J \\ 0 & \text{for all vertices from which t is reachable by the edges in } J \\ 1 & \text{for all other vertices} \end{cases} \tag{2.24}$$

Now given an optimal solution $\bar{\pi}$ to the DRP_{spath} the θ can be calculated as:

$$\theta = \min_{\substack{e \notin J \\ \text{such that} \\ \pi_u - \pi_v > 0}} \{c_e - \{\pi_u - \pi_v\}\} \tag{2.25}$$

In calculation of θ the denominator is always 1 because according to the equation 2.24 for any $e = (u,v) \notin J$ such that $\bar{\pi}_u - \bar{\pi}_v > 0$ the difference $\bar{\pi}_u - \bar{\pi}_v$ is always 1. Now the improved solution to the D_{spath} is:

$$\pi_{new} = \pi + \theta\bar{\pi} \tag{2.26}$$

The process of improving the dual solution π, computing the set J accordingly and resolving the DRP_{spath} is repeated until an optimal objective function value of $\bar{\pi}_s - \bar{\pi}_t = 0$ is obtained. At this point we have an optimal solution to both P_{spath} and D_{spath}.
In other words the *primal-dual method* reduces the *shortest path problem* to the repeated solution of simpler reachability problem[2].

[2]Given a directed graph $G = (V,E)$ the reachability problem finds out the set of nodes reachable from a given node $v \in V$.

Chapter 3

Related Research

3.1 Introduction

In this chapter we discuss the *maximum cardinality matching problem* and two variations of this problem known as the *maximum fractional matching problem* and the *unconstrained fractional matching problem*. Unlike the *maximum cardinality matching problem* the *maximum fractional matching problem* allows fractional values on the edges of a graph instance and the *unconstrained fractional matching* problem allows arbitrary capacities on the vertices and negative values on the edges of a graph instance.

In Section 3.2 we discuss the *maximum cardinality matching problem* on both the bipartite and the general graph instances and the related research works. In Section 3.3 we discuss the *maximum fractional matching* problem and the related research works and in Section 3.4 we discuss the *unconstrained fractional matching* problem and the related research works.

3.2 Maximum Cardinality Matching (MCMP)

Given a graph $G = (V, E)$, a matching in G is a subset of edges ($M \subseteq E$) such that no two edges in M share a vertex $v \in V$. *The maximum cardinality matching problem* (MCMP) is to find a matching M of maximum size.

Given a matching M in G, a vertex $v \in V$ is called saturated with respect to M if v has an incident edge $e \in M$, otherwise v is called unsaturated with respect to M. A matching M

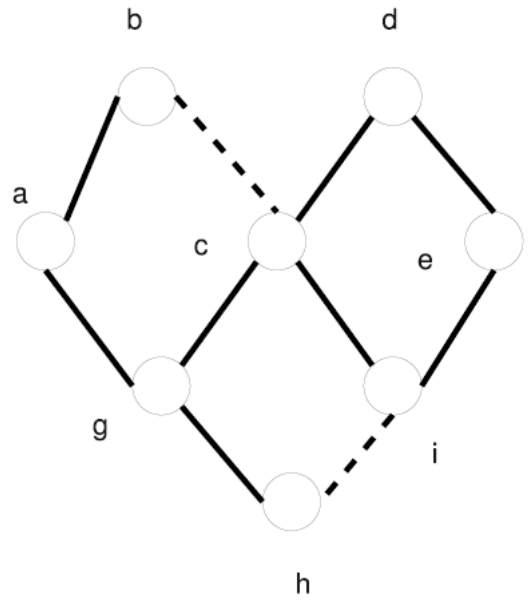


Figure 3.1: Matching Example

is called perfect if all the vertices in V are saturated with respect to M. A path P is called an *alternating path* with respect to M if it is simple[1] and consists of edges $e \in E - M$ and $e \in M$ alternately. An *alternating path* P is called an *augmenting path* with respect to M if P starts and ends at unsaturated vertices and consists of odd number of edges.

In Figure 3.1 $M = \{(b,c),(h,i)\}$ is a matching, vertex a is unsaturated and vertex b is saturated with respect to M. The alternating paths $\{a,b,c,d\}$ and $\{g,h,i,e\}$ are two augmenting paths with respect to M.

The following theorem is one of the most important theorem in matching theory known as the Berge's theorem [3]. This theorem gives the foundation for an efficient matching algorithm(see [16, Ch. 19] for the proof).

Theorem 3.2.1. *(Berge's theorem) A matching M in a graph G is maximum if and only if*

[1] A path that has no repeated vertices

there exists no augmenting path with respect to M.

3.2.1 Cardinality Matching In Bipartite Graphs

A graph $G = (V, E)$ is bipartite if its vertex set V can be partitioned into two disjoint sets U and V such that no edge $e \in E$ has both endpoints in the same set of bipartition. In a word a graph G is called bipartite if it contains no odd length cycle. So a bipartite graph can be denoted as $G_B = (U \bigcup V, E)$, where $(u, v) \in E$ implies $u \in U$ and $v \in V$.

In 1916 König showed that each regular[2] bipartite graph has a perfect matching. In 1931 König characterized the maximum size of a matching in bipartite graph through his matching theorem. The theorem states that for any bipartite graph the maximum size of a matching is equal to the minimum size of a vertex cover[3] [23] .

One method to find the maximum matching in bipartite graph using flows is as follows [2, ch. 17] [11]:

- Direct all edges in G_B from U to V and assign a capacity of 1 to all edges.
- Attach a vertex s with directed edge (s, u_i) of infinite capacity to all vertices $u_i \in U$ and a vertex t with directed edge (v_j, t) of infinite capacity to all vertices $v_j \in V$.
- Apply one of the maximum flow algorithms to find a maximum flow in this modified graph.

If the algorithm due to Malhotra, Pramodh-Kumar and Maheshwari (this algorithm is also known as the MPM's algorithm [16]) for the maximum flow problem [20] is used then the

[2] A graph $G = (V, E)$ is called regular if all the vertices $v \in V$ have the same degree.

[3] Given a graph $G = (V, E)$ a vertex cover is a set C of vertices such that all edges $e \in E$ are incident to at least one vertex in C.

maximum matching can be computed in $O(|V|^3)$ time.

Another algorithm to find out maximum matching in a bipartite graph is due to Hopcroft and Karp [10]. Starting with an empty matching M in G_B this algorithm proceeds in phases. In each phase it carries out an alternating breadth first search procedure starting at all unsaturated vertices with respect to M in U in order to find at least one unsaturated vertex with respect to M in V and thus constructs a directed acyclic graph known as the Hungarian Tree. The Hungarian tree contains all non-matching[4] edges in its odd levels and matching[5] edges in its even levels. Then the algorithm finds a maximal set of vertex disjoint augmenting paths of same minimum length k and augments M along these paths simultaneously. It can be shown that there are at most $\sqrt{|V|}$ phases while each phase costs $O(|E|)$. So the total running time of this algorithm is $O(|E||V|^{1/2})$. This algorithm is regarded as the fastest algorithm for finding maximum matching in bipartite graphs.

In 2004 M. Mucha and Piotr Sankowski [21] showed that the running time of maximum bipartite matching can be theoretically improved to $O(|V|^{2.38})$.

3.2.2 Cardinality Matching in General Graphs

The fundamental goal to find a maximum cardinality matching in a graph G is to find a M-augmenting path and increase the size of matching M. But it is difficult to apply alternating breadth first search for finding a maximum matching described in the previous section directly to the general graphs. The difficulty arise due to the existence of odd length cycles consisting of alternate matching and non-matching edges within augmenting paths

[4] An edge e is called non-matching if $e \notin M$

[5] An edge e is called matching if $e \in M$

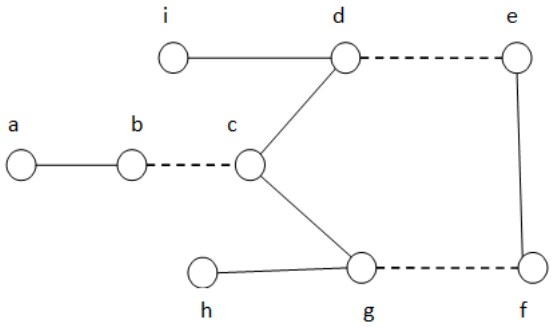


Figure 3.2: Example of a blossom

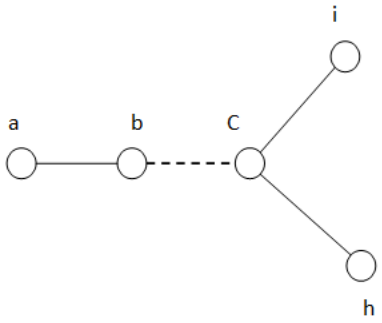


Figure 3.3: Example of a shrunken blossom

in a general graph. If we try to find an augmenting path with respect to a matching M in a general graph $G = (V,E)$ the odd length cycle within an augmenting path can cause the alternating breadth first search to give false results. Such odd length cycle is called a *blossom*. In Figure 3.2 the odd length cycle c,d,e,f,g,c is a blossom and c is called the base of the blossom.

In 1965 Edmonds proposed a method of shrinking[6] the blossom into a supervertex and continuing the search for an augmenting path in the resulting graph. Figure 3.3 shows the shrunken form. In the same paper Edmonds proposed an algorithm for finding maximum

[6]Shrinking a blossom means deletion of its vertices and connecting all their incident edges to the supervertex

cardinality matching in general graph which runs in $O(|V|^4)$ time [9]. In 1994 Vazirani improved the running time to $O(|E||V|^{1/2})$ by using the technique of searching for a maximal set of shortest augmenting paths in $\sqrt{V}$ phases [26].

3.2.3 Integer Program for MCMP

Given a graph $G = (V,E)$ the MCMP can be formulated as following integer program:

$$\begin{aligned} &\text{maximize} && \sum_{e \in E} q_e && && \text{(MCMP)} \\ &\text{subject to} && \sum_{e \in E(v)} q_e \leq 1 && \forall v \in V \\ & && q_e = \{0,1\} && \forall e \in E \end{aligned}$$

Here $E(v)$ is the set of edges incident on $v \in V$ and q_e is the binary decision variable associated with each edge $e \in E$. If q_e is equal to 1 then the corresponding edge e is in matching. Such a formulation as MCMP is known as $0-1$ *integer program*. The general version of *0-1 integer program* is NP-complete [13]. Allowing q_e to take any real value in MCMP allow us to apply some interesting techniques to approximately solve this problem.

3.3 Maximum Fractional Matching

Given a graph $G = (V,E)$ the *fractional matching problem* (FMP) is the linear programming relaxation[7] of the integer program MCMP. The FMP allows the edges $e \in E$ to take any

[7] The program obtained after the removal of the integrality constraints

value in between 0 and 1. The problem is formulated as the following linear program:

$$
\begin{aligned}
&\text{maximize} && \sum_{e \in E} q_e && && \text{(FMP)}\\
&\text{subject to} && \sum_{e \in E(v)} q_e \leq 1 && \forall v \in V\\
& && q_e \leq 1 && \forall e \in E\\
& && q_e \geq 0 && \forall e \in E
\end{aligned}
$$

If G is bipartite than the size of the maximum cardinality matching is equal to the value of the maximum fractional matching in G. This is true due to the fact that G contains no odd length cycle and the vertex-edge incidence matrix of G is totally unimodular (see [24, Ch. 2]). Therefore the linear program above returns an integral solution.

In 1989 Bourjolly and Pulleyblank showed that, given a general graph an optimal solution to the fractional matching problem is half integral, that is $q_e \in \{0, 1/2, 1\}$ for every edge e and this solution can be constructed in $O(|E||V|)$ time [5] .

3.4 Unconstrained Fractional Matching Problem (UFMP)

The *unconstrained fractional matching problem* (UFMP) is a generalization of the *fractional matching problem*. Given a graph $G = (V, E)$ with arbitrary positive integer capacity c_v defined on every vertex $v \in V$, the UFMP allows each edge $e \in E$ to take any value in $[-\infty, +\infty]$. In 2009 Bobby Chan [6] in his Masters thesis proposed a combinatorial algorithm for this problem using the primal-dual approach. The running time complexity of the algorithm is $O(|V|^2|E|)$. In this section we study his approach.

3.4.1 Problem formulation

The *unconstrained fractional matching problem* is given by the following LP formulation:

$$\begin{aligned} &\text{maximize} && \sum_{e \in E} q_e && && \text{(UFMP)} \\ &\text{subject to} && \sum_{e \in E(v)} q_e \leq c_v && \forall v \in V \\ & && q_e \leqslant 0 && \forall e \in E \end{aligned}$$

Here q_e is the decision variable associated with each edge $e \in E$ and $E(v)$ is the set of edges incident on a vertex $v \in V$.

Considering the UFMP as the dual problem the primal problem is defined as:

$$\begin{aligned} &\text{minimize} && \sum_{v \in V} c_v y_v && && (P_{\text{ufm}}) \\ &\text{subject to} && y_u + y_v = 1 && \forall e = (u,v) \in E \\ & && y_v \geq 0 && \forall v \in V \end{aligned}$$

Here c_v is the capacity of the vertex $v \in V$ and y_v is the decision variable associated with v. Given a feasible solution q to the dual (UFMP), the set J of tight dual constraints is defined as:

$$J = \left\{ v \mid c_v = \sum_{e \in E(v)} q_e \right\} \tag{3.1}$$

From the set J the restricted primal is formulated as:

$$
\begin{aligned}
&\text{minimize} && \sum_{e \in E} a_e && && (RP_{\text{ufm}})\\
&\text{subject to} && y_u + y_v + a_e = 1 && \forall e = (u,v) \in E\\
& && y_u \geq 0 && \forall u \in J\\
& && y_u = 0 && \forall u \notin J\\
& && a_e \geq 0 && \forall e \in E
\end{aligned}
$$

where a_e is the artificial variable introduced for every edge $e \in E$ in order to gain feasibility in the primal. From the restricted primal the dual of the restricted primal is derived as:

$$
\begin{aligned}
&\text{maximize} && \sum_{e \in E} q_e && && (DRP_{\text{ufm}})\\
&\text{subject to} && \sum_{e \in E(v)} q_e \leq 0 && \forall v \in J\\
& && q_e \lesseqgtr 0 && \forall e \in E\\
& && q_e \leq 1 && \forall e \in E
\end{aligned}
$$

The feasible and optimal solutions to the DRP_{ufm} are denoted by $\bar{q}$ and $\bar{q}_{\text{opt}}$ respectively. The multiplying factor θ required to improve the dual solution q without violating the feasibility is chosen as:

$$
\theta = \min_{\substack{v \notin J \\ \text{such that} \\ \sum_{e \in E(v)} \bar{q}\text{opt}_e > 0}} \left\lfloor \frac{c_v - \sum_{e \in E(v)} q_e}{\sum_{e \in E(v)} \bar{q}\text{opt}_e} \right\rfloor \tag{3.2}
$$

Instead of solving the RP_{ufm} using simplex method as mentioned in the general primal-

dual method (see Section 2.6) the optimal solution to the DRP_{ufm} is characterized using the ideas due to Krishnamurti *et al.* [17] and a phased algorithm is proposed to solve the DRP_{ufm} optimally as described in the next sections.

3.4.2 Characterization of the optimal solution to the DRP_{ufm}

In an attempt to characterize the optimal solution to the DRP_{ufm} the following definitions are introduced:

- Given a feasible solution q to the UFMP a vertex $v \in V$ is called saturated with respect to q if $\sum_{e \in E(v)} q_e = c_v$. Otherwise, u is called unsaturated with respect to q. In other words a vertex $v \in V$ is called saturated if it is a member of set J.

- Given a feasible solution $\bar{q}$ to the DRP_{ufm} any edge $e \in E$ is called saturated with respect to $\bar{q}$ if $\bar{q}_e = 1$. Otherwise e is called unsaturated with respect to $\bar{q}$.

- Given a feasible solution q to the UFMP and a feasible solution $\bar{q}$ to the DRP_{ufm} an *alternating walk* $P_{\text{alt}}(v_1, e_1, v_2, e_2, v_3, \ldots, e_{n-1}, v_n)$ with respect to $\bar{q}$ is a vertex-edge sequence where the vertices v_1 and v_n are unsaturated and the vertices $v_2, v_3, \ldots, v_{n-1}$ are saturated with respect to q, the edges $e_1, e_3, \ldots$ are unsaturated with respect to $\bar{q}$. P_{alt} is called an *alternating path* if $v_1, v_2, \ldots, v_n$ are pairwise distinct, otherwise P_{alt} is called an *alternating lasso*. If P_{alt} is a alternating path of odd length then it is called an *augmenting path*. Otherwise if P_{alt} is a alternating lasso of odd length then it is called an *augmenting lasso*.

Theorem 3.4.1. *(Chan 2009) A feasible solution $\bar{q}$ to the DRP_{ufm} is optimal if and only if there does not exist any augmenting path or lasso with respect to $\bar{q}$.*

Given any feasible solution $\bar{q}$ to a DRP_{ufm} instance the optimal solution $\bar{q}_{\text{opt}}$ is found by searching for augmenting paths or lassos with respect to $\bar{q}$ and augmenting the $\bar{q}$ value along them until any augmenting path or lasso exists.

Augmentation of current $\bar{q}$ value by δ along an augmenting walk is carried out by adding $+\delta$ to the odd indexed edges called the plus edges and $-\delta$ along the even indexed edges called the minus edges. This augmentation process increased the overall $\bar{q}$ value by δ. The intermediate vertices along the augmenting path remains saturated even after the augmentation process because adding and subtracting same amount δ from two incident edges of a vertex v do not change the sum of the edge values incident on v. To maintain feasibility the value of δ is calculated as:

$$\delta = min\left(\frac{1-q_{e_{2i-1}}}{k_{2i-1}}\right) \qquad for\ i = 1,2,\ldots,n \tag{3.3}$$

Here k_{2i-1} denotes the number of times the edge e_{2i-1} is visited along an augmenting walk. δ is strictly positive as all the odd indexed edges contain a value strictly less than 1 according to the definition of the augmenting path/lasso.

3.4.3 Algorithm for solving DRP_{ufm}

A phased technique [6] similar to the Hopcroft and Karp's approach for the *maximum bipartite matching* [10] is used for solving the DRP_{ufm} optimally. The phased technique is only applicable to the bipartite graphs. So if the input graph is a general graph the first step of this approach is to convert the general graph to its bipartite equivalent. Next we describe the reduction to bipartite graph.

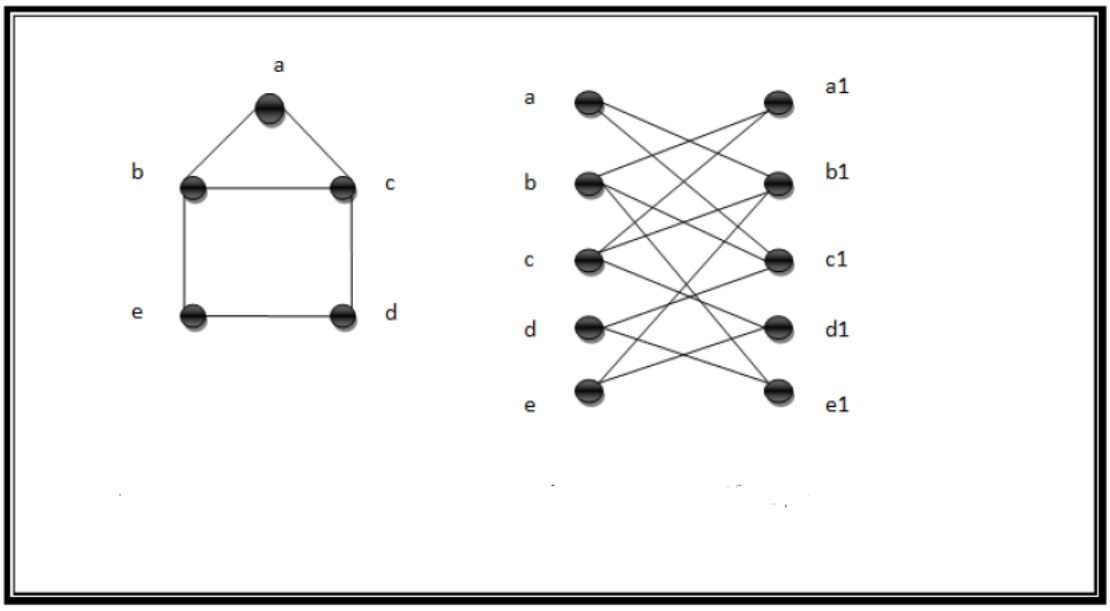


Figure 3.4: A general graph and its bipartite equivalent

Bipartite Graph Reduction

Given a general graph $G = (V,E)$ the bipartite graph $G_B = (U \bigcup V, E)$ is constructed as follows:

- For each vertex $v \in V$ in G two vertices $u_1 \in U$ and $v_1 \in V$ in G_B are defined and the capacity of each of these vertices are defined as $c_{u_1} = c_{v_1} = c_v$.
- For each edge $e = (u,v) \in E$ two edges $e_1 = (u_1, v_2) \in E$ and $e_2 = (u_2, v_1) \in E$ are defined.

The set of feasible values q assigned on the edges of G_B has a one-to-one mapping to the set of feasible values q^g assigned on the edges of G.

Lemma 3.4.1. *(Chan 2009) Given an optimal solution q^g_{opt} to UFMP in graph G with objective function value $\sum_{e \in E} q^g_{opt_e} = Q$ there must exist an optimal solution q to UFMP in*

G_B with objective function value $\sum_{e \in E} q_{opt_e} = 2Q$

Proof. Given a feasible solution q in G_B for each pair of edges $(u_1, v_2), (u_2, v_1) \in E$ in G_B, the corresponding edge $(u, v) \in E$ in G can be assigned the value:

$$q^g_{(u,v)} = \frac{1}{2} q_{(u_1,v_2)} + \frac{1}{2} q_{(u_2,v_1)} \tag{3.4}$$

As q is a feasible solution so for each pair of vertex $u_1 \in U$ and $v_1 \in V$ in G_B,

$$\sum_{e \in E(u_1)} q_e \leq c_{u_1} \tag{3.5}$$

$$\sum_{e \in E(v_1)} q_e \leq c_{v_1} \tag{3.6}$$

Adding two equations results in,

$$\sum_{e \in E(u_1)} q_e + \sum_{e \in E(v_1)} q_e \leq c_{u_1} + c_{v_1} \tag{3.7}$$

Since the capacities c_{u_1} and c_{v_1} of the vertices $u_1 \in U$ and $v_1 \in V$ in G_B is equal to the capacity c_v of the corresponding vertex $v \in V$ in G, the equation 3.7 can be expanded as:

$$\begin{aligned} q_{(u_1,v_2)} + q_{(u_1,v_3)} + \ldots + q_{(u_1,v_k)} + q_{(u_2,v_1)} + q_{(u_3,v_1)} + \ldots + q_{(u_k,v_1)} &\leq 2c_v \\ = \frac{1}{2}(q_{(u_1,v_2)} + q_{(u_1,v_3)} + \ldots + q_{(u_1,v_k)} + q_{(u_2,v_1)} + q_{(u_3,v_1)} + \ldots + q_{(u_k,v_1)}) &\leq c_v \end{aligned}$$

According to equation 3.4 we have:

$$\sum_{e \in E(v)} q^g_e \leq c_v \tag{3.8}$$

So the conversion from q to q^g maintains feasibility.

Since according to equation 3.4 any feasible value $q_{(u_1,v_2)}$, $q_{(u_2,v_1)}$ for each pair of edges $(u_1,v_2),(u_2,v_1) \in E$ in G_B contribute $\frac{1}{2}$ to the feasible value $q^g_{(u,v)}$ of the corresponding edge $(u,v) \in E$ in G so for an optimal UFMP solution q_{opt} in G_B with optimal objective function value $\sum_{e \in E} q_{\text{opt}_e} = Q$, the corresponding general graph G will have an optimal solution q^g_{opt} with optimal objective function value $\sum_{e \in E} q^g_{opt_e} = Q/2$.

Again given a feasible solution q^g to a general graph G for each edge $(u,v) \in E$, the corresponding edges $(u_1,v_2),(u_2,v_1) \in E$ in G_B can be assigned the value:

$$q_{(u_1,v_2)} = q_{(u_2,v_1)} = q^g_{(u,v)} \tag{3.9}$$

As $c_{u_1} = c_{v_1} = c_v$ and q^g is a feasible solution so for each pair of vertices $u_1 \in U$ and $v_1 \in V$

$$\sum_{e \in E(u_1)} q_e \leq c_{u_1} \text{and} \tag{3.10}$$

$$\sum_{e \in E(v_1)} q_e \leq c_{v_1} \tag{3.11}$$

So the conversion from q_g to q also maintains feasibility.

From equation 3.9 it is clear that if for an optimal solution q^g_{opt} in G the objective function value $\sum_{e \in E} q^g_{\text{opt}_e} = Q$ then the corresponding bipartite graph G_B has an optimal solution q_{opt} with objective function value $\sum_{e \in E} q_{\text{opt}_e} = 2Q$ □

The conversion from the general graph G to the bipartite graph G_B does not change the running time complexity of the phased algorithm since it only increases the number of vertices and edges by a factor of 2.

Phased Algorithm for Solving DRP_{ufm}

Given a bipartite graph $G_B = (U \bigcup V, E)$ and a set of saturated vertices J with respect to a feasible solution q to UFMP the phased algorithm works as follows:

Starting with a feasible DRP_{ufm} solution $\bar{q} = 0$ the algorithm divides the search for augmenting paths with respect to $\bar{q}$ in phases. In each phase a maximal set of plus edge disjoint augmenting paths of same length are discovered and augmented simultaneously. The algorithm stops when no augmenting path is found with respect to $\bar{q}$. For the correctness of the procedure the reader is referred to [6].

The procedure described above is carried out in two steps:

1. Starting with all unsaturated vertices in U an alternating breadth first search is deployed until an unsaturated vertex in V is found. From the vertices at even level $2k$ the vertices at odd level $2k+1$ are obtained by following the edges with $\bar{q}_e < 1$ (plus edges) that have not been visited yet and from the vertices at odd level $2k+1$ the vertices at even level $2k+2$ are obtained by following the edges that have not been visited yet. Actually during this alternating breadth first search all the vertices obtained at even levels are in U and all the vertices obtained at odd levels are in V. If an unsaturated vertex is found at an odd level $2k+1$ the search process stops after obtaining all the unsaturated vertices at that level and the next step is carried out. If no unsaturated vertex is found on any odd level it means that the current solution to the DRP_{ufm} is optimal.

2. After the alternating breadth first search process an augmentation and topological delete process is deployed on the plus edges. Starting with an unsaturated vertex at the last level, a path P is traced back to the first level. This path is an augmenting path and along this path the $\bar{q}$ value is augmented by the amount δ as described in equation 3.3. After this augmentation all the plus edges along the augmenting paths

become saturated. Then all these plus edges are deleted. If all the incoming edges of a vertex are deleted then that vertex is deleted too and deletion of a vertex causes all its outgoing edges to be deleted. The process continues until there are no more paths to trace back to the first level.

The above two steps are repeated until no augmenting path with respect to $\bar{q}$ is found by the alternating breadth first search. The number of phases is bounded by the following lemma:

Lemma 3.4.2. *(Chan 2009) The length of the augmenting paths found in each successive phase is strictly increasing.*

The length of the augmenting paths can be at most $|V|$ so the number of phases in the phased algorithm can be at most $|V|$. Each phase takes $O(|E|)$ time. So the running time of the phased algorithm is $O(|E||V|)$. We refer the reader to the thesis of Bobby Chan [6] for details of the proof.

3.4.4 Primal-Dual Algorithm for UFMP

The proposed primal-dual algorithm for the UFMP works as follows:
If the input graph contains odd length cycle then it is converted to its bipartite equivalent. Starting with a feasible UFMP solution q the set of tight UFMP constraints i.e. the set of tight vertices J is formed. Using the set J the current DRP_{ufm} instance is formed and the optimal solution $\bar{q}_{\text{opt}}$ is obtained using the phased algorithm. If the objective function value for the DRP_{ufm} solution is zero than the algorithm stops and returns the optimal solution q_{opt} to the $UFMP$. Otherwise θ is calculated using the equation 3.2 and the current dual solution q is improved as:

$$q = q + \theta \bar{q}_{\text{opt}} \tag{3.12}$$

Then the algorithm iterates with formation of the set J with respect to new q.

The number of iteration of the primal-dual algorithm is bounded using the following two facts:

- The vertices that enter set J never come out of set J. The fact is true because during the augmentation process any vertex in set J is used as the intermediate vertex along the augmenting path and the same amount δ is added to and subtracted from two incident edges of J causing it to remain saturated.

- Each θ calculation guarantees at least one vertex enters set J at the beginning of next iteration.

The above two facts ensure that the primal-dual algorithm iterates $|V|$ times. As each DRP iteration requires $O(|E||V|)$ time so the total running time of the primal-dual algorithm is $O(|V|^2|E|)$.

Chapter 4

Maximum Charge Problem

4.1 Introduction

In this chapter, we study the *maximum charge problem* with capacity constraints. For an undirected graph instance $G = (V, E)$ the *maximum charge problem* allows for arbitrary positive capacities assigned on both the vertices and the edges. Unlike the *unconstrained fractional matching problem* this problem has capacity constraints on the edge value q_e for all $e \in E$ for any undirected graph instance $G = (V, E)$.

In Section 4.3 we present the linear programming formulation for the *maximum charge problem*. In Section 4.4 we present the primal-dual interpretation for the *maximum charge problem*. In Section 4.5 we present the characterization of the optimal solution to the dual of the restricted primal (DRP) for the problem and a combinatorial approach for solving the DRP optimally. In Section 4.6 we discuss an example for the problem that illustrates the need for working with shortest paths to compute the optimal solution to the DRP. Last of all in Section 4.7 we discuss the analysis of the *primal-dual algorithm* for the problem.

4.2 Definitions

In this section we define some terms to be used in the next sections. The notation is from Krishnamurti *et al.* [17]. All the definitions are with respect to a graph $G = (V, E)$.

Walk: A walk S is a vertex-edge sequence $\{v_1, e_1, v_2, e_2, v_3, \ldots, v_l, e_l, v_{l+1}\}$ of length l where the same vertex and same edge may occur more than once with different subscripts.

Path: A walk S is called a path P if and only if the vertices $\{v_1, v_2, v_3, \ldots, v_l, v_{l+1}\}$ are pairwise distinct.

Lasso: A walk is called a lasso L if and only if the vertices $\{v_1, v_2, v_3, \ldots, v_l, v_{l+1}\}$ and the edges $\{e_1, e_2, e_3, \ldots, e_l\}$ are pairwise distinct and $v_{l+1} = v_n$ for $n \in \{1, 2, \ldots, l\}$. Actually L is a combination of a path P and a cycle C. L degenarates to a cycle if $v_n = v_1$. The length of the lasso L is $l = p + c$ where p is the length of the path and c is the length of the cycle.

Extended Lasso: A walk is called an extended lasso L' if it consists of a path $P = \{v_1, e_1, v_2, e_2, v_3, \ldots, v_{n-1}, e_{n-1}, v_n\}$, a cycle $C = \{v_n, e_n, v_{n+1}, e_{n+1}, v_{n+2}, \ldots, v_l, e_l, v_n\}$ and path $P^{-1} = \{v_n, e_{n-1}, v_{n-1}, \ldots, v_3, e_2, v_2, e_1, v_1\}$. Stated otherwise $L' = PCP^{-1} = LP^{-1}$. If the length of the lasso L and the path P is l and p respectively then the length of L' is $l' = l + p = 2p + c = 2l - c$

4.3 Problem Formulation

Given an undirected graph $G = (V, E)$ with positive capacities c_v and c_e assigned to each vertex $v \in V$ and each edge $e \in E$ respectively the *maximum charge problem* seeks to maximize the assignment of feasible charges to the edges $e \in E$ such that the total sum of the charges is maximized. A feasible charge q is a function defined on E such that for each edge $e \in E$, $0 \leq q_e \leq c_e$ and for each vertex $v \in V$, $0 \leq \sum_{e \in E(v)} q_e \leq c_v$, where $E(v)$ is the set of edges incident on v. Formally the *maximum charge problem* is given by the following

linear program :

$$
\begin{aligned}
&\text{maximize} && \sum_{e \in E} q_e && && \text{(MCP)}\\
&\text{subject to} && \sum_{e \in E(v)} q_e \leq c_v && \forall v \in V\\
& && q_e \leq c_e && \forall e \in E\\
& && q_e \geq 0 && \forall e \in E
\end{aligned}
$$

The MCP formulation is a generalization of the UFMP formulation in Section (3.4). The difference is that, in UFMP the decision variable $q_e, \forall e \in E$ is unconstrained i.e. it can take any positive or negative value in the range $[-\infty, +\infty]$, whereas in the MCP formulation the charge $q_e, \forall e \in E$ is constrained to take a value in the range $[0, c_e]$.

4.4 Primal-Dual interpretation

We consider the MCP as the dual and define the corresponding primal linear program as:

$$
\begin{aligned}
&\text{minimize} && \sum_{v \in V} c_v y_v + \sum_{e \in E} c_e z_e && && (P_{\text{mcp}})\\
&\text{subject to} && y_u + y_v + z_e \geq 1 && \forall e = (u, v) \in E\\
& && y_u \geq 0 && \forall u \in V\\
& && z_e \geq 0 && \forall e \in E
\end{aligned}
$$

where y_u and y_v are the decision variables associated with the vertices u and v respectively and z_e is the decision variable associated with the edge e.

By introducing a surplus variable w_e for every edge we write the P_{mcp} in standard form as:

$$\begin{aligned}
&\text{minimize} && \sum_{v\in V} c_v y_v + \sum_{e\in E} c_e z_e && && (PS_{\text{mcp}})\\
&\text{subject to} && y_u + y_v + z_e - w_e = 1 && \forall e=(u,v)\in E\\
& && y_u \geq 0 && \forall u \in V\\
& && z_e \geq 0 && \forall e \in E\\
& && w_e \geq 0 && \forall e \in E
\end{aligned}$$

Since the primal (PS_{mcp}) constraints are satisfied at equality by any feasible solution to the primal, the *primal complementary slackness* conditions will be satisfied automatically by the optimal solutions to the primal and the dual.

The dual complementary slackness conditions are:

$$(c_v - \sum_{e\in E(v)} q_e) y_v = 0 \qquad \forall v \in V \tag{4.1}$$

$$(c_e - q_e) z_e = 0 \qquad \forall e \in E \tag{4.2}$$

$$q_e w_e = 0 \qquad \forall e \in E \tag{4.3}$$

Given any feasible solution to the dual (MCP), the goal is to obtain a feasible solution in the primal (PS_{mcp}) satisfying the *dual complementary slackness conditions* above. For any feasible solution q to the dual (MCP), the sets J_1, J_2 and J_3 of the tight dual constraints are:

$$J_1 = \left\{ v \mid c_v = \sum_{e\in E(v)} q_e \right\},\ J_2 = \{e \mid q_e = c_e\},\ J_3 = \{e \mid q_e = 0\}$$

From the sets J_1, J_2 and J_3 the restricted primal can be formulated as:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} a_e && && (RP_{\text{mcp}}) \\
\text{subject to} \quad & y_u + y_v + z_e - w_e + a_e = 1 && \forall e = (u,v) \in E \\
& y_u \geq 0 && \forall u \in J_1 \\
& y_u = 0 && \forall u \notin J_1 \\
& z_e \geq 0 && \forall e \in J_2 \\
& z_e = 0 && \forall e \notin J_2 \\
& w_e \geq 0 && \forall e \in J_3 \\
& w_e = 0 && \forall e \notin J_3 \\
& a_e \geq 0 && \forall e \in E
\end{aligned}
$$

where a_e is the artificial variable introduced for every edge $e \in E$ in order to gain feasibility in the primal. Traditionally we can use the ordinary *simplex* method to solve the RP_{mcp} (see [22, Ch. 5]). If the objective function value ζ for the optimal solution to the RP_{mcp} is zero then we have an optimal pair of solutions to the PS_{mcp} and MCP as they satisfy the *complementary slackness conditions*. If $\zeta > 0$ then we try to improve the value of the MCP solution by taking a linear combination of the current MCP solution and the optimal

solution to the dual of the restricted primal (DRP) defined below:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} q_e && && (DRP_{\text{mcp}}) \\
\text{subject to} \quad & \sum_{e \in E(v)} q_e \leq 0 && \forall v \in J_1 \\
& q_e \leq 0 && \forall e \in J_2 \\
& q_e \geq 0 && \forall e \in J_3 \\
& q_e \leq 1 && \forall e \in E
\end{aligned}
$$

Let $\bar{q}_{opt}$ be the optimal solution to the DRP_{mcp}. The improved solution to the MCP is given by:

$$q^* = q + \theta \bar{q}_{opt} \tag{4.4}$$

Now we need to choose an appropriate θ in order to maintain feasibility in *MCP* and we have:

$$\theta_1 = \min_{\substack{e \notin J2 \\ \text{such that} \\ \bar{q}_{opt_e} > 0}} \left[\frac{c_e - q_e}{\bar{q}_{opt_e}} \right] \tag{4.5}$$

$$\theta_2 = \min_{\substack{e \notin J3 \\ \text{such that} \\ (-1)\bar{q}_{opt_e} > 0}} \left[\frac{q_e}{(-1)\bar{q}_{opt_e}} \right] \tag{4.6}$$

$$\theta_3 = \min_{\substack{v \notin J1 \\ \text{such that} \\ \sum_{e \in E(v)} \bar{q}_{opt_e} > 0}} \left[\frac{c_v - \sum_{e \in E(v)} q_e}{\sum_{e \in E(v)} \bar{q}_{opt_e}} \right] \tag{4.7}$$

$$\theta = \min\{\theta_1, \theta_2, \theta_3\} \tag{4.8}$$

4.5 Primal-Dual Algorithm

Our goal here is to give a combinatorial algorithm for solving DRP_{mcp}, this in turn would imply a polynomial time primal-dual algorithm for the *maximum charge problem* with capacity constraints. At first, we characterize the optimal solution to the DRP_{mcp} and then propose two approaches to solve it optimally.

4.5.1 Characterization of the Optimal Solution to the DRP_{mcp}

The DRP_{mcp} is in some sense a simplified version of the original dual (MCP). The only striking difference is that, the edge values can be negative in the DRP_{mcp} solutions, whereas for the MCP they are constrained to be positive. In the DRP_{mcp}, vertices in the set J_1 have a capacity of zero, edges in the set J_2 have capacity zero, edges not in the set J_2 have capacity one, edges in the set J_3 can accquire only positive values and all other edges can acquire either positive or negative values. However, it is still possible to characterize the optimal solution to the DRP_{mcp} using the proof technique that characterizes the optimal solution to the MCP due to Krishnamurti *et al.* [17]. This proof is substantially similar to the proof in [6].

We begin with a few definitions. The definitions below are with respect to a feasible solution $\bar{q}$ to the DRP_{mcp}.

- The capacity $\bar{c_e}$ for each edge $e \in E$ is 0 if $e \in J_2$, 1 otherwise.
- An edge is called full if $\bar{q}_e = 1$.
- An edge is called tight if $\bar{q}_e = 0$.
- An edge e is called saturated if $e \notin J_2$ and full ($\bar{q}_e = 1$) or if $e \in J_2$ and tight ($\bar{q}_e = 0$).

Otherwise e is called unsaturated.

- The capacity $\bar{c_v}$ for each vertex $v \in V$ is 0 if $v \in J_1$, ∞ otherwise. A vertex v is called saturated if $\bar{c_v} = \sum_{e \in E(v)} \bar{q}_e$ where $E(v)$ is the set of edges incident on v. Otherwise v is called unsaturated.

- An alternating walk S is a sequence of vertices and edges, $v_1, e_1, v_2, e_2, v_3, \ldots, v_{n-1}, e_{n-1}, v_n$ such that the first and last vertices are unsaturated, intermediate vertices are saturated, the odd indexed edges $e_1, e_3, \ldots$ are unsaturated and even indexed edges $e_2, e_4, \ldots$ are not in J_3.

- An alternating walk of odd length is called an augmenting walk.

- An augmenting walk is called an augmenting path if $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$ are pairwise distinct, otherwise it is called an augmenting lasso.

- Augmenting a walk by δ means adding $+\delta$ to the odd indexed edges and $-\delta$ to the even indexed edges along the walk.

The following theorem characterizes the necessary and sufficient condition for the optimality of a DRP_{mcp} solution :

Theorem 4.5.1. *$\bar{q}$ is not an optimal solution to the DRP_{mcp} if and only if there exist an augmenting path or an augmenting lasso with respect to $\bar{q}$.*

Proof. Suppose that, there exists an augmenting walk P of length l with respect to $\bar{q}$. If P is an augmenting path then all the vertices and edges along P are pairwise distinct. Let $\delta = min\{(\bar{c}_{e_1} - \bar{q}_{e_1}), (\bar{c}_{e_3} - \bar{q}_{e_3}), \ldots, (\bar{c}_{e_l} - \bar{q}_{e_l})\}$. Clearly $\delta > 0$ as all odd indexed edges are unsaturated. We can add δ to any odd indexed edges and subtract δ from any even indexed edges along P without violating any edge capacity constraint.
But if P is an extended lasso then an edge can occur maximum twice along P. So we can

set,

$\delta = \frac{1}{2} min\{(\bar{c}_{e_1} - \bar{q}_{e_1}), (\bar{c}_{e_3} - \bar{q}_{e_3}), \ldots, (\bar{c}_{e_l} - \bar{q}_{e_l})\}$. In this case also $\delta > 0$. Then we can add δ to any odd indexed edges and subtract δ from any even indexed edges along P without violating feasibility.

In both cases the resulting solution maintains feasibility and has larger objective function value. So $\bar{q}$ is not optimal.

Again, let us assume that, $\bar{p}$ be a feasible solution that is not optimal, $\bar{q}$ is an optimal solution to the DRP_{mcp} with strictly large value and among all optimal solutions $\bar{q}$ minimizes $\sum_{e \in E} |\bar{p}_e - \bar{q}_e|$. We use the proof technique similar to [17]. As $\bar{q}$ has larger value, there exists a vertex v_1 such that

$$\sum_{e \in E(v_1)} \bar{p}_e < \sum_{e \in E(v_1)} \bar{q}_e$$

Commence a walk starting at v_1 as follows:

- Odd Step : Assume that the vertex v_{2i+1} is reached along an edge with $\bar{p}_e > \bar{q}_e$. If v_{2i+1} is not saturated in $\bar{q}$ then stop. Otherwise among all outgoing edges $e_{2i+1} = (v_{2i+1}, v_{2i+2})$ there must exists an edge with $\bar{p}_e < \bar{q}_e$. Take this edge and perform the Even Step.

- Even Step : Assume that the vertex v_{2i+2} is reached along an edge with $\bar{p}_e < \bar{q}_e$. If v_{2i+2} is not saturated in $\bar{p}$ then stop. Otherwise among all outgoing edges $e_{2i+2} = (v_{2i+2}, v_{2i+3})$ there must exists an edge with $\bar{p}_e > \bar{q}_e$. Take this edge and perform the Odd Step.

If the walk terminates with even number of edges, starting with a $\bar{p}$ unsaturated vertex and ending with a $\bar{q}$ unsaturated vertex then $\bar{q}$ does not minimize $\sum_{e \in E} |\bar{p}_e - \bar{q}_e|$. Therefore the walk ends with odd number of edges, starting and ending with $\bar{p}$ unsaturated vertices. We can add δ to the odd indexed edges and subtract δ from even indexed edges without

violating feasibility for a suitably chosen δ and increase the value of the objective function, therefore the walk is an augmenting walk which is either an augmenting path or an augmenting lasso. □

Note that if the input graph is bipartite than the augmenting walk is an augmenting path and we can easily claim that there always exists an integral optimal solution to the DRP_{mcp}. Because if there exists an augmenting path, then there exists an integral augmenting path. Moreover along an augmenting path no edge occurs more than once. So the odd indexed edges can be assigned an additional value of 1 and even indexed edges can be assigned an additional value of -1 without violating any constraints. Thus if $\bar{q}$ is a integral feasible solution then augmenting the solution along the integral augmenting path always gives an integral feasible solution.

4.5.2 Phased Approach for Solving the DRP_{mcp}

In this section we present a technique for solving the DRP_{mcp} optimally. We call this approach the *phased approach*. This approach is reminiscent of the Hopcraft and Karp's approach for the *unweighted bipartite matching* [10]. Given a bipartite graph $G_B = (U \bigcup V, E)$, a feasible charge q and the associated instance of DRP_{mcp} this approach searches for a maximal set of augmenting paths of shortest length k with respect to a feasible DRP_{mcp} solution $\bar{q}$ phase by phase and augment them simultaneously. We define *phase* to be a step that finds a maximal set of augmenting paths of same length k, such that there does not exist any augmenting path of length strictly less than k. This approach starts with a feasible solution of $\bar{q} = 0$ to the current instance of DRP_{mcp} and stops when no augmenting path is found with respect to $\bar{q}$. Each phase involves the following two steps:

- Constructing a Hungarian Tree (HT) with respect to the current feasible DRP_{mcp}

solution $\bar{q}$ by deploying an alternating breadth first search.

- Discovering the same length augmenting paths in the HT and augmenting $\bar{q}$ along the augmenting paths.

In the next sections we provide the detailed description of the steps outlined.

4.5.3 Construction of the Hungarian Tree (HT)

Let us consider a bipartite graph instance $G_B = (U \bigcup V, E)$, a feasible charge q and the associated instance of DRP_{mcp}. Let $\bar{q}$ be a feasible solution to the current instance of DRP_{mcp}. We construct the Hungarian Tree (HT) as follows:

Starting with all the unsaturated vertices with respect to $\bar{q}$ in U, we do an alternating breadth first search until an unsaturated vertex with respect to $\bar{q}$ in V is encountered. At level 1 we include all the unsaturated vertices in U and mark them as visited so that these vertices can't be used in any other levels. From the vertices at level 1 we obtain the vertices in V at level 2 by following the edges that are unsaturated with respect to $\bar{q}$, i.e. the edges that are either not full ($\bar{q}_e \neq 1$) if not in set J_2 or not tight ($\bar{q}_e \neq 0$) if in set J_2. We mark all the vertices at level 2 as visited and then check whether any of these vertices are unsaturated with respect to $\bar{q}$. If at least one unsaturated vertex is found in level 2 then we stop. Otherwise from the vertices at level 2 we obtain the unmarked vertices in U at level 3 following the edges not in J_3 and these vertices are saturated with respect to $\bar{q}$. In this way from the vertices in U at any odd level $2l-1$ ($l = 1,2,\ldots,$) we obtain the vertices in V at the even level $2l$ following the unsaturated edges with respect to $\bar{q}$ and from the vertices in V at the even level $2l$ we obtain the vertices in U at the odd level $2l+1$ following the edges not in J_3.
The alternating breadth first search process continues until we come up with an even level

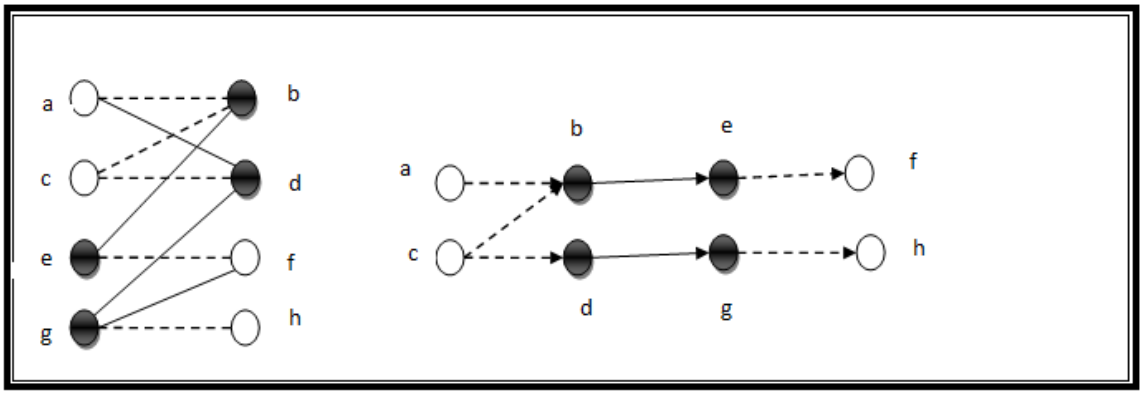


Figure 4.1: A bipartite graph instance and the associated HT

containing at least one unsaturated vertex with respect to $\bar{q}$. At this point the construction of the HT is completed.

Figure 4.1 shows a bipartite graph instance and the HT constructed from it using the alternating breadth first search. Here the dark circles represent the saturated vertices and dotted lines represent the unsaturated edges with respect to $\bar{q}$ and solid lines represent the edges not in J3.

Algorithm 4.1 outlines the construction process of the HT. From the construction process of HT it is clear that the edges on the odd levels have spare capacity and the value of $\bar{q}$ along these edges can be augmented by $+1$, similarly value of $\bar{q}$ along the edges on the even levels can be decremented by -1 without affecting the feasibility of the solution. For this reason we call the edges at odd level as *positive edges* and the edges at the even level as *negative edges*. It is always possible to augment the value of $\bar{q}$ along an augmenting path such that at least one *positive edge* is saturated (for the edge, $\bar{q}_e = 1$ if not in J_2 or $\bar{q}_e = 0$ if in J_2).

In the following lemma we will argue that we can safely delete the saturated edges from

Algorithm 4.1 Construction of the Hungarian Tree

Require: A bipartite graph $G_B = (U \bigcup V, E_B)$ with feasible charge q, the associated instance of DRP_{mcp} and a feasible solution $\bar{q}$ to the current instance of DRP_{mcp}

Ensure: A Hungarian Tree

1: Define level 1, $L_1 \leftarrow$ all unsaturated vertices $u_{unsat} \in U$
2: Define level 2, $L_2 \leftarrow$ the vertices $v \in V$ obtained following the unsaturated edges $e_{unsat} \in E$ from $u_{unsat} \in L_1$
3: Mark all vertices $u \in L_1$ and $v \in L_2$ as visited
4: Set $j \leftarrow 1$
5: **while** L_{j+1} contains no unsaturated vertex and not empty **do** {Here L_{j+1} *means level*(j+1)}
6: Set $j \leftarrow j+2$
7: Define level j, $L_j \leftarrow$ the vertices $u \in U$ are not visited and obtained following the edges $e \notin J_3$ from $v \in L_{j-1}$
8: Define level $j+1$, $L_{j+1} \leftarrow$ the vertices $v \in V$ are not visited and obtained following the unsaturated edges $e_{unsat} \in E$ from $u \in L_j$
9: Mark all vertices $u \in L_j$ and $v \in L_{j+1}$ as visited
10: **end while**

the HT as the saturated edges would not be involved in any other augmenting paths in the same HT.

Lemma 4.5.1. *In a phase if an edge is saturated as a result of some augmentation then it stays saturated during the phase.*

Proof. Let e be the edge that is first saturated by path p_1 and then unsaturated by path p_2 in the same phase. Note that p_1 and p_2 have the same length. There exists a path, in the symmetric difference of these two paths with length shorter than the length of any path in the phase violating the assumption that all the augmenting paths in a phase are of same length. □

Lemma 4.5.1 is tight in the sense that if an edge is not saturated then it can be part of some other augmenting paths in the same phase. Indeed a *positive edge* with $\bar{q}_e = -\delta$ can belong to $\delta + 1$ augmenting paths in a phase as shown in figure 4.2. In this figure all the vertices

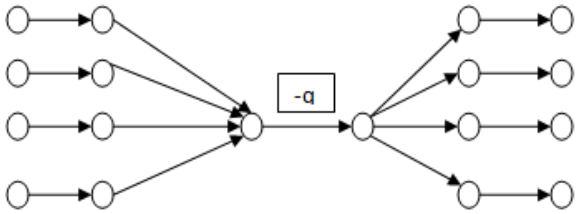


Figure 4.2: An unsaturated edge shared by multiple augmenting paths

except the leftmost and rightmost vertices are saturated and all the edges except the middle one has $\bar{q}_e = 0$. There are multiple augmenting paths sharing a positive edge.

4.5.4 Topological Erase Approach for Finding the Augmenting Paths

Given a HT the maximal set of same length augmenting paths in each phase can be found using an approach similar to the topological plus edge delete process due to Bobby Chan in the phased approach for the unconstrained fractional matching problem [6]. This approach is based on the Hopcroft and Karp's idea [10]. In Section 3.4.3 we have provided a brief discussion of the approach. In this Section we discuss how we use this approach for finding the augmenting paths in the HT and augment them simultaneously.

Given a HT we begin at an unsaturated vertex at the last level of the HT and trace a path back to the start level. This path is an augmenting path. We add the path to a set S. We augment the current DRP_{mcp} solution $\bar{q}$ along the augmenting path by adding +1 to the positive edges and adding -1 to the negative edges. At least one positive edge along the path becomes saturated due to this augmentation process. Then we delete all the saturated edges along the augmenting path.

As the HT is a directed acyclic graph, for each vertex residing on any level other than the start level we can keep track of the vertices of its previous level from which it has an incoming edge using a list called the parent list. For each vertex we define the first element of the parent list as its current parent. Whenever an incoming edge of a vertex is deleted along the augmenting path we change the current parent to the next element of the parent list and delete the previous parent from the list. If the parent list of a vertex become empty due to the edge deletions, we delete that vertex and all its outgoing edges. The augmentation and the saturated edge deletion process is repeated as long as there exists an augmenting path to trace back from the last level to the start level of HT. When the process stops there are no more augmenting paths in the HT. Algorithm 4.2 outlines the *topological erase approach*.

Algorithm 4.2 Topological Erase Algorithm

Require: A Hungarian Tree
Ensure: Maximal set of augmenting paths of the same length
1: Set $V_{last} \leftarrow$ All unsaturated vertices at the last level of HT
2: **while** V_{last} is not empty **do**
3: Trace a path back to the start level
4: Add the path to the set of paths in the phase
5: Do the augmentation process along the path
6: Delete the saturated edges
7: If a vertex has no incoming edge then delete the vertex too
8: When a vertex is deleted, outgoing edges are deleted too
9: **end while**

Augmenting the DRP_{mcp} value along an augmenting path requires k steps, where k is the length of the augmenting paths. If all the positive edges on the augmenting paths are saturated as a result of the augmentation (and for all augmentations) then the total work required is $O(|E|)$, as each edge is deleted once. However in the worst case only a single

positive edge may be saturated and the total work required in each phase would be $O(k|E|)$.

4.5.5 Blocking Charge Approach for Finding the Augmenting Paths

In this section we explore a way to perform a set of augmentations in parallel such that all the augmenting paths of length k in a *phase* are destroyed. The idea is similar to the idea of computing *blocking flows* in the MPM's algorithm due to Malhotra *et al.* for the *network flow problem* [20]. In this section we use *charge* as the synonym of the feasible charge $\bar{q}$ to the DRP_{mcp}.

Given a charge $\bar{q}$ and another charge $\bar{q}'$, we define $\bar{q}'$ as *k–blocking charge* if $\bar{q}+\bar{q}'$ is feasible and there do not exist augmenting path of length at most k in the HT. Next we study how to obtain a blocking charge given a HT using an approach similar to that of MPM's algorithm. The Figure 4.3 outlines the *blocking charge approach*.

Given a feasible charge $\bar{q}$ and a HT, we define capacity c_e^{ht} for every edge in the HT. For the positive edges we define the capacity $c_e^{ht} = \begin{cases} 1-\bar{q}_e : e \notin J_2 \\ 0-\bar{q}_e : e \in J_2 \end{cases}$. For the negative edges we define the capacity c_e^{ht} as unbounded. For every vertex v in the HT we define total incoming capacity as `incapacity(v)` $= \sum_{e \in E^-(v)} c_e^{ht}$ and total outgoing capacity as `outcapacity(v)` $= \sum_{e \in E^+(v)} c_e^{ht}$. Here E^- and E^+ are the sets of incoming and outgoing edges respectively. We compute the deficit of each vertex v residing in the levels other than

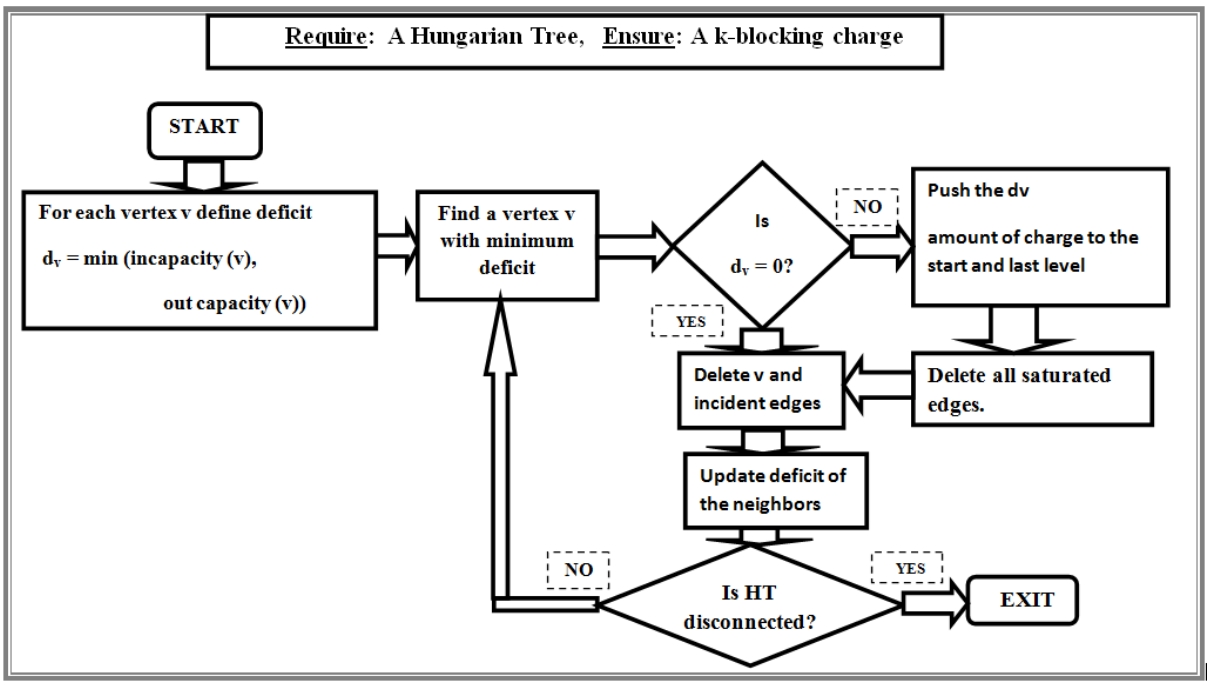


Figure 4.3: Blocking charge approach

the start and last level of HT as:

$$d_v = min\{\texttt{incapacity(v)}, \texttt{outcapacity(v)}\} \tag{4.9}$$

For each vertex v in the start level we compute the deficit as:

$$d_v = \texttt{outcapacity(v)} \tag{4.10}$$

And For each vertex v in the last level we compute the deficit as:

$$d_v = \texttt{incapacity(v)} \tag{4.11}$$

Next we search for the vertex with minimum deficit in the HT. Let u be the vertex with minimum deficit d_u. We consider one of the following three cases for the minimum deficit:

Case 1: If $d_u = 0$ we delete u and all the incident edges. We then update the deficit of the neighbors of u and again search for the vertex with minimum deficit.

Case 2: If u is at even level, we carry out the following steps:

1. We distribute $+d_u$ amount of charge among the edges coming into u in the following way:

   - We define $\texttt{push}_{in} = d_u$.
   - We pick the first edge e coming into u . We add $+1$ amount of charge to the edge and update $\texttt{push}_{in} = \texttt{push}_{in} - 1$. We include the other end point v of the edge e to the list of neighbors in previous level of u. We denote this list as $\texttt{neighbor}_P$.
   - We continue to pick the edges coming into u one by one and add $+1$ amount of charge to the edges and update $\texttt{push}_{in}$ as mentioned in the previous step until $\texttt{push}_{in} = 0$.

2. We then distribute $-d_u$ amount of charge among the outgoing edges of u in the following way:

   - We define $\texttt{push}_{out} = -d_u$
   - As u is at even level all its outgoing edges are negative edges and the capacities of these edges are unbounded. So we pick the first edge going out of u and add $\texttt{push}_{out}$ amount of charge to the edge and update $\texttt{push}_{out} = 0$. We include the other end point of the edge to the list of neighbors in next level of u. We denote this list as $\texttt{neighbor}_N$.

Case 3: If u is at odd level, we carry out the following steps:

1. We distribute $-d_u$ amount of charge among the edges coming into u in the following way:

    - We define $\texttt{push}_{in} = -d_u$
    - As u is at odd level all its incoming edges are negative edges and the capacities of these edges are unbounded. So we pick the first edge coming into u and add $\texttt{push}_{in}$ amount of charge to the edge and update $\texttt{push}_{in} = 0$. We include the other end point of the edge to the list of neighbors in the previous level of u. We denote this list as $\texttt{neighbor}_P$.

2. We then distribute $+d_u$ amount of charge among the edges going out of u in the following way:

    - We define $\texttt{push}_{out} = d_u$.
    - We pick the first edge e going out of u . We add $+1$ amount of charge to the edge and update $\texttt{push}_{out} = \texttt{push}_{out} - 1$. We include the other end point of the edge to the list of neighbors in next level of u. We denote this list as $\texttt{neighbor}_N$.
    - We continue to pick the edges going out of u one by one and add amount of charge to the edges and update $\texttt{push}_{out}$ as mentioned in the previous step until $\texttt{push}_{out} = 0$.

Again if u is at the start level (level 1) we only need to distribute $+d_u$ amount of charge to the edges going out of u or if u is at last level (last level of HT is always be an even level) we only need to distribute $+d_u$ amount of charge to the edges coming into u.

After this update the vertices that are distance 1 away from u will have some excess charge[1] either positive or negative. As every other vertex has deficit at least the deficit of u and the excess charge is no more than the deficit of u, therefore it is possible to push this excess charge towards both the first level and last level.

We push the excess charge towards the start level in the following way :

1. We define $\texttt{CurrentLevel} = \texttt{level}_u - 1$. We repeat the next three steps until $\texttt{CurrentLevel} = \texttt{StartLevel}$
2. We define the list $\texttt{ConsiderableNodes} = \texttt{neighbor}_P$.
3. From the list ConsiderableNodes we pop the first vertex v. We find out the amount of excess charge for v. If CurrentLevel is a odd level we distribute the excess amount of charge along its incoming edges using the step 1 of *Case 3*. Else if CurrentLevel is even level we distribute the excess amount of charge along its incoming edges using the step 1 of *Case 2*. We continue to pop the vertices from the list ConsiderableNodes one by one and distribute the excess amount of charge to their incoming edges until the list ConsiderableNodes is empty.
4. We then update the $\texttt{CurrentLevel} = \texttt{CurrentLevel} - 1$.

We push the excess charge towards the last level in the following way :

1. We define $\texttt{CurrentLevel} = \texttt{level}_u + 1$. We repeat the next three steps until $\texttt{CurrentLevel} = \texttt{LastLevel}$

[1] If for any intermediate vertex total incoming charge is not equal to total outgoing charge then the difference between these two charge amount is called the excess charge for that vertex

2. We define the list `ConsiderableNodes` $=$ `neighbor`$_N$.

3. From the list `ConsiderableNodes` we pop the first vertex v. We find out the amount of excess charge for v. If `CurrentLevel` is a odd level we distribute the excess amount of charge along its outgoing edges using the step 2 of *Case 3*. Else if `CurrentLevel` is even level we distribute the excess amount of charge along its outgoing edges using the step 2 of *Case 2*. We continue to pop the vertices from the list `ConsiderableNodes` one by one and distribute the excess amount of charge to their outgoing edges until the list `ConsiderableNodes` is empty.

4. We then update the `CurrentLevel` $=$ `CurrentLevel` $+ 1$.

While pushing the charge we ensure that at most one edge become partially saturated and other edges are either empty or completely saturated. Once an edge is saturated it is deleted from the HT. Thus for vertex u either all the incoming or the outgoing edges are saturated and deleted from the HT, therefore the vertex u is deleted and the deficits of its neighbors are updated. We repeat the whole process as long as the last level is reachable from the first level of the HT. Algorithm 4.3 provides an outline for computing *blocking charge* in a HT.

Lemma 4.5.2. *Given a Hungarian Tree a blocking charge can be computed using Algorithm 4.3 in $O(|E|)$ time.*

Proof. In the computation of blocking charge the total time required to identify a vertex with minimum deficit is $O(|E|)$. Total time required to delete all saturated edges is $O(|E|)$. Therefore the total time to partially saturate all the edges is $O(|E|)$. So the total time required for computing a blocking charge is $O(|E|)$. Hence the claim. □

Algorithm 4.3 Computing Blocking Charge

Require: A Hungarian Tree and a feasible DRP_{mcp} solution called charge $\bar{q}$
Ensure: A blocking charge in HT

1: **for** $e \in HT$ **do**
2: **if** e is in odd level **then**
3: the capacity $c_e^{ht} = \begin{cases} 1 - \bar{q}_e : e \notin J_2 \\ 0 - \bar{q}_e : e \in J_2 \end{cases}$
4: **else**
5: The capacity c_e^{ht}= unbounded
6: **end if**
7: **end for**
8: **for** $v \in HT$ **do**
9: The deficit $d_v = min\{$`incapacity(v), outcapacity(v)`$\}$
10: `incharge`$[v] = 0$
11: `outcharge`$[v] = 0$
12: **end for**
13: **repeat**
14: Find a vertex u with minimum deficit d_u
15: **if** $d_u = 0$ **then**
16: Delete u and its all incident edges and update the deficits of neighboring vertices
17: **else**
18: **if** `level`$_u \neq$ `StartLevel` **then**
19: `Push-excesscharge-to-the-start-level`$(u, level_u, d_u)$
20: **end if**
21: **if** `level`$_u \neq$ `LastLevel` **then**
22: `Push-excesscharge-to-the-last-level`$(u, level_u, d_u)$
23: **end if**
24: Delete all the saturated edges from the HT
25: Delete u and all its remaining incident edges
26: Update the deficit of all the vertices in HT
27: **end if**
28: **until** HT becomes disconnected

Procedure 4.4 Push-excesscharge-to-the-start-level(u, $level_u$, d_u)

1: CurrentLevel $=$ level$_u$, excesscharge $= d_u$, neighbor$_P = \phi$
2: Push-excesscharge-to-previous-level(u, CurrentLevel, excesscharge, neighbor$_P$)
3: CurrentLevel $=$ CurrentLevel $- 1$
4: **while** CurrentLevel $\neq$ StartLevel **do**
5: ConsiderableNodes $=$ neighbor$_P$, neighbor$_P = \phi$
6: **while** ConsiderableNodes $\neq \phi$ **do**
7: $v =$ ConsiderableNodes.$pop(0)$
8: excesscharge $=$ abs(outcharge[v] - incharge[v])
9: Push-excesscharge-to-previous-level(v, CurrentLevel, excesscharge, neighbor$_P$)
10: **end while**
11: CurrentLevel $=$ CurrentLevel $- 1$
12: **end while**

Push-excesscharge-to-previous-level(u, CurrentLevel, excesscharge, neighbor$_P$)

1: PreviousLevel $=$ CurrentLevel $- 1$
2: **if** CurrentLevel is odd level **then**
3: push$_{in} = -$excesscharge
4: **for** $w \in$ PreviousLevel **do**
5: **if** (w,u) is an edge **then**
6: $\bar{q}_{(w,u)} = \bar{q}_{(w,u)} + push_{in}$
7: outcharge$[w]$ $=$ outcharge$[w] + \bar{q}_{(w,u)}$
8: incharge$[u]$ $=$ outcharge$[w] + \bar{q}_{(w,u)}$
9: neighbor$_P$ $=$ neighbor$_P + w$
10: push$_{in} = 0$
11: break
12: **end if**
13: **end for**
14: **else**
15: push$_{in}$ $=$ excesscharge
16: **for** $w \in$ PreviousLevel **do**
17: **if** (w,u) is an edge **then**
18: $\bar{q}_{(w,u)} = \bar{q}_{(w,u)} + 1$
19: push$_{in}$ $=$ push$_{in} - 1$
20: $outcharge[w] = outcharge[w] + \bar{q}_{(w,u)}$
21: $incharge[u] = outcharge[w] + \bar{q}_{(w,u)}$
22: neighbor$_P$ $=$ neighbor$_P + w$
23: **if** push$_{in} = 0$ **then**
24: break
25: **end if**
26: **end if**
27: **end for**
28: **end if**

Procedure 4.5 Push-excesscharge-to-the-last-level(u, $level_u$, d_u)

1: CurrentLevel $=$ level$_u$, excesscharge $= d_u$, neighbor$_N = \phi$
2: Push-excesscharge-to-next-level(u,CurrentLevel,excesscharge,neighbor$_N$)
3: CurrentLevel $=$ CurrentLevel $+ 1$
4: **while** CurrentLevel $\neq$ LastLevel **do**
5: ConsiderableNodes $=$ neighbor$_N$, neighbor$_N = \phi$
6: **while** ConsiderableNodes $\neq \phi$ **do**
7: $v =$ ConsiderableNodes.$pop(0)$
8: excesscharge $=$ abs(outcharge[v] - incharge[v])
9: Push-excesscharge-to-next-level(v, CurrentLevel, excesscharge, neighbor$_N$)
10: **end while**
11: CurrentLevel $=$ CurrentLevel $+ 1$
12: **end while**

Push-excesscharge-to-next-level(u, CurrentLevel, excesscharge, neighbor$_N$)

1: NextLevel $=$ CurrentLevel $+ 1$
2: **if** CurrentLevel is even level **then**
3: push$_{out} = -$excesscharge
4: **for** $w \in$ NextLevel **do**
5: **if** (u,w) is an edge **then**
6: $\bar{q}_{(u,w)} = \bar{q}_{(u,w)} +$ push$_{out}$
7: outcharge$[u]$ $=$ outcharge$[u] + \bar{q}_{(u,w)}$
8: incharge$[w]$ $=$ incharge$[w] + \bar{q}_{(u,w)}$
9: neighbor$_N$ $=$ neighbor$_N + w$
10: push$_{out} = 0$
11: break
12: **end if**
13: **end for**
14: **else**
15: push$_{out}$ $=$ excesscharge
16: **for** $w \in$ NextLevel **do**
17: **if** (u,w) is an edge **then**
18: $\bar{q}_{(u,w)} = \bar{q}_{(u,w)} + 1$
19: push$_{out}$ $=$ push$_{out} - 1$
20: $outcharge[u] = outcharge[u] + \bar{q}_{(u,w)}$
21: $incharge[w] = incharge[w] + \bar{q}_{(u,w)}$
22: neighbor$_N$ $=$ neighbor$_N + w$
23: **if** push$_{out} = 0$ **then**
24: break
25: **end if**
26: **end if**
27: **end for**
28: **end if**

4.5.6 Analysis of the Phased Approach

The Algorithm 4.6 provides an outline of our proposed phased approach for solving an instance of DRP_{mcp} optimally. Now in order to bound the number of phases for Algo-

Algorithm 4.6 Phased Algorithm

Require: A bipartite graph $G_B = (U \bigcup V, E)$ with feasible charge q and the associated instance of DRP_{mcp}

Ensure: Optimal solution to the current instance of DRP_{mcp}

1: Start with a feasible DRP_{mcp} solution $\bar{q} = 0$
2: find-optimal = *No*
3: **repeat**
4: Construct the HT using Algorithm 4.1
5: **if** HT = ϕ **then**
6: find-optimal = *Yes*
7: **else**
8: Find a blocking charge in the HT using Algorithm 4.3
9: **end if**
10: **until** find-optimal = *Yes*
11: **return** $\bar{q}$

rithm 4.6 we will show that the length of the augmenting paths found in each successive phase is strictly increasing. We will discuss the idea in the following lemma.

Lemma 4.5.3. *The length of the set of minimum length augmenting paths discovered in each successive phase (line 3) of Algorithm 4.6 is strictly increasing.*

Proof. Suppose that in phase i we augment the DRP_{mcp} solution $\bar{q}_1$ along a maximal set S of augmenting paths of shortest length k and obtain the DRP_{mcp} solution $\bar{q}_2$. Let us consider any augmenting path p' with respect to $\bar{q}_2$ at the phase $i+1$. The length of p' must be greater than k. Otherwise the assumptions that S is maximal and k is the shortest length, will be violated.

Again let p' share a saturated edge e_s with any augmenting path p in S then e_s has to be

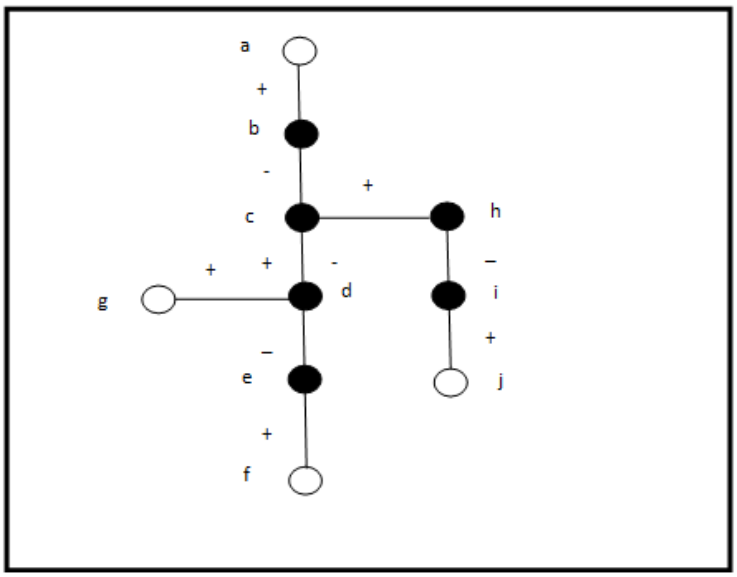


Figure 4.4: An edge shared by two augmenting paths

used as a negative edge in p' although it has been used as a positive edge in p. So the length of p' must be greater than the length of p. Otherwise if we consider the symmetric difference of the paths p and p' we come up with a path with length less than k violating the assumption that k was of shortest length. The situation is illustrated in Figure (4.4). Here $p = \{a,b,c,d,e,f\}$ and $p' = \{g,d,c,h,i,j\}$. Both p and p' share the saturated edge $e_s = (c,d)$. In p, e_s is used as a positive edge and thus become saturated due to augmentation whereas in p' e_s is used as a negative edge. Now let us consider the symmetric difference of p and p'. We have two augmenting paths $\{a,b,c,h,i,j\}$ and $\{g,d,e,f\}$ in the symmetric difference and the second one has length 3. The claim follows. □

Theorem 4.5.2. *Algorithm 4.6 can compute an optimal solution to the* DRP_{mcp} *in* $O(|E||V|)$ *time.*

Proof. Each phase of Algorithm 4.6 consists of two steps. First one is the construction of the HT and second one is the computation of *blocking charge* in the HT. The HT is constructed by carrying out an alternating breadth first search on the current DRP_{mcp} instance.

The HT can be constructed in $O(|E|)$ time since no edges will be visited more than once during the alternating breadth first search process. According to Lemma 4.5.2 the *blocking charge* in a HT can be computed in $O(|E|)$ time. So total time required by each phase is $O(|E|)$.

According to the Lemma 4.5.3, the length of the augmenting paths increases with each phase. The length of the augmenting paths can be at most $|V|$. So the number of phases is bounded by $O(|V|)$.

So the total running time of the phased algorithm is bounded by $O(|E||V|)$.

□

4.6 Why choose the shortest augmenting paths

In this section we discuss the reason for choosing shortest augmenting paths. We show with an example that choosing any augmenting path other than the shortest one may cause the total number of DRP_{opt} iterations to be unbounded.

Consider the bipartite graph in Figure 4.5. The charges assigned on the edges and the capacities are as follows:

- all the edges have capacity 1.
- all the vertices except v and u have capacity 1.
- the vertices u and v have capacity 2ε.
- all the vertices except the four endpoints are in set J_1

Let us denote the charge solution by q. Consider the sets of tight dual constraints J_1, J_2, J_3 and the DRP_{mcp} formulation. For this example, J_1 contains all vertices except the four endpoints and $J_2 = J_3 = \phi$. There can be two optimal solutions to the associated DRP_{mcp}

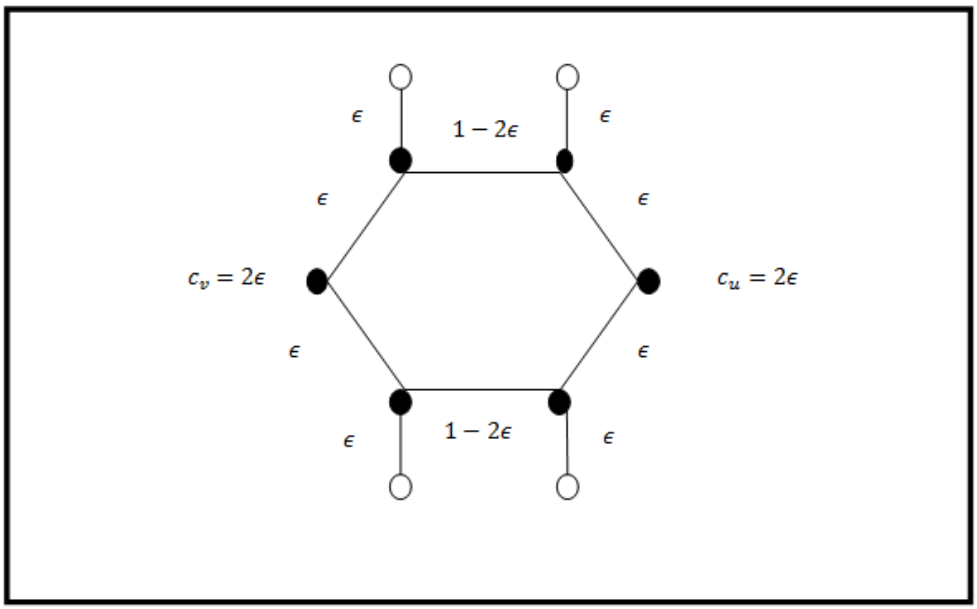


Figure 4.5: A bipartite graph

instance. The solutions are shown in Figure 4.6. Both optimal solutions have the objective function value 2. The Solution:1 chooses the shortest augmenting paths of length three each. Let us consider that, our algorithm discovers the Solution:2 that highlights the augmenting paths of length five each. We denote the solution by $\bar{q}$. Then $q+\theta\bar{q}$ has to satisfy all the constraints in the *MCP* to maintain feasibility.

Consider the new charge solution value for the edge (x,y) shown in the Solution:2 of Figure 4.6. We have, $\varepsilon-\theta 1 \geq 0$ which implies $\theta \leq \varepsilon$. Therefore the new charge solution after update to $q+\theta\bar{q}$ is shown in Figure 4.7.

In the next iteration the optimal solution to the DRP_{mcp} is shown in Figure 4.8. Now consider the new charge value for the edge (y,z) in Figure 4.8. We have $2\varepsilon-\theta 1 \geq 0$ which implies $\theta \leq 2\varepsilon$. Therefore the new charge solution is shown in Figure 4.9.

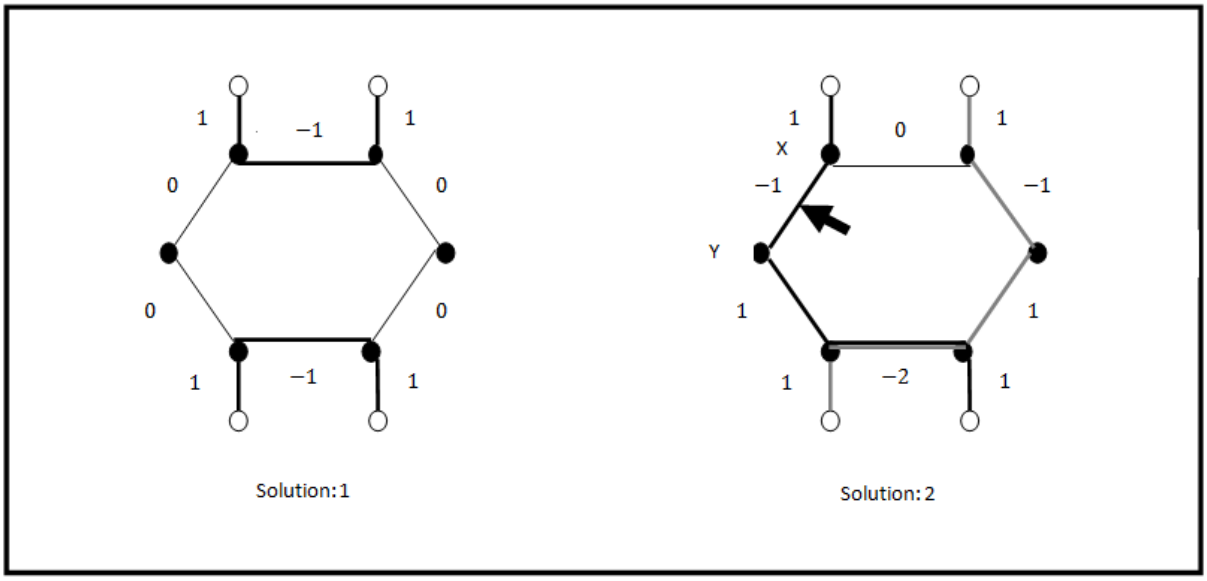


Figure 4.6: Two optimal solution to the DRP_{mcp}

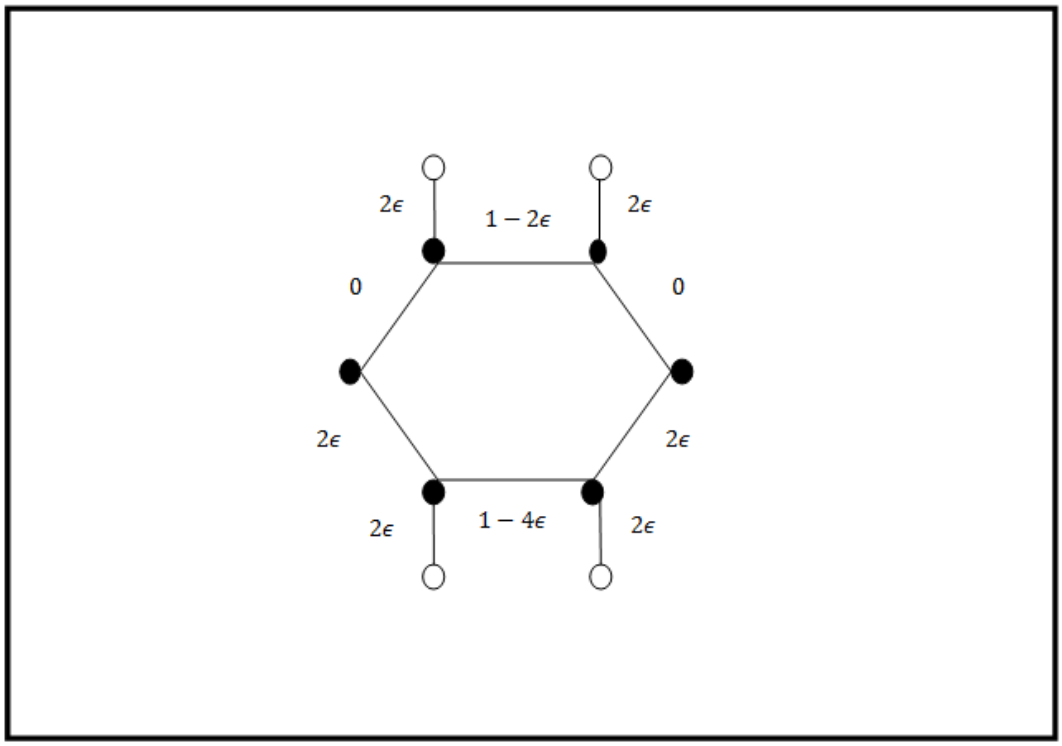


Figure 4.7: New charge solution $q + \varepsilon\bar{q}$

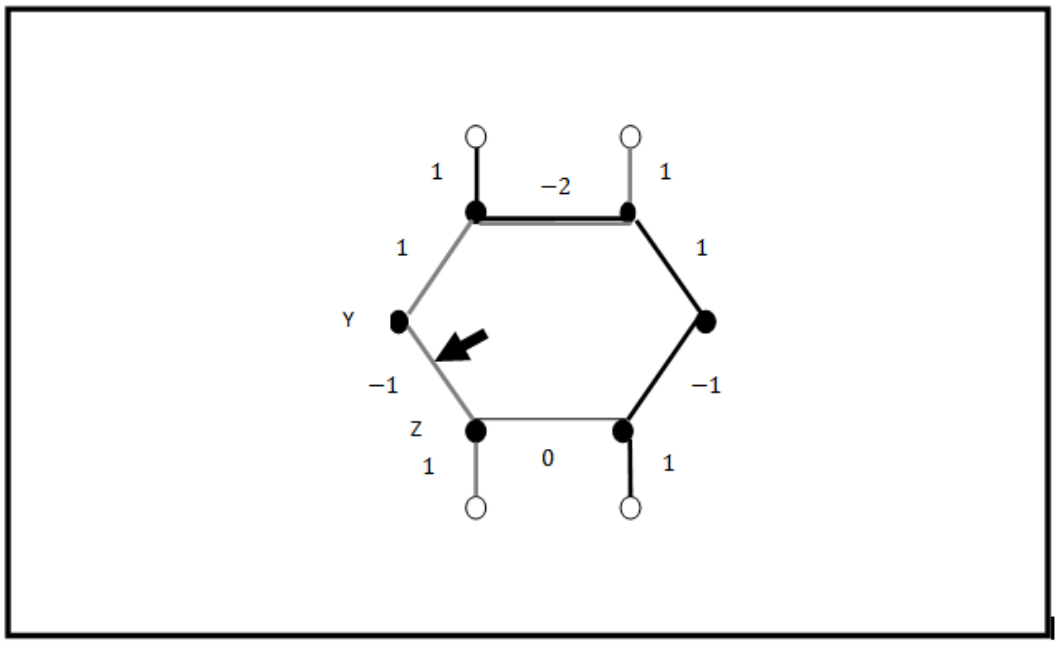


Figure 4.8: Optimal solution to the current DRP_{mcp} instance

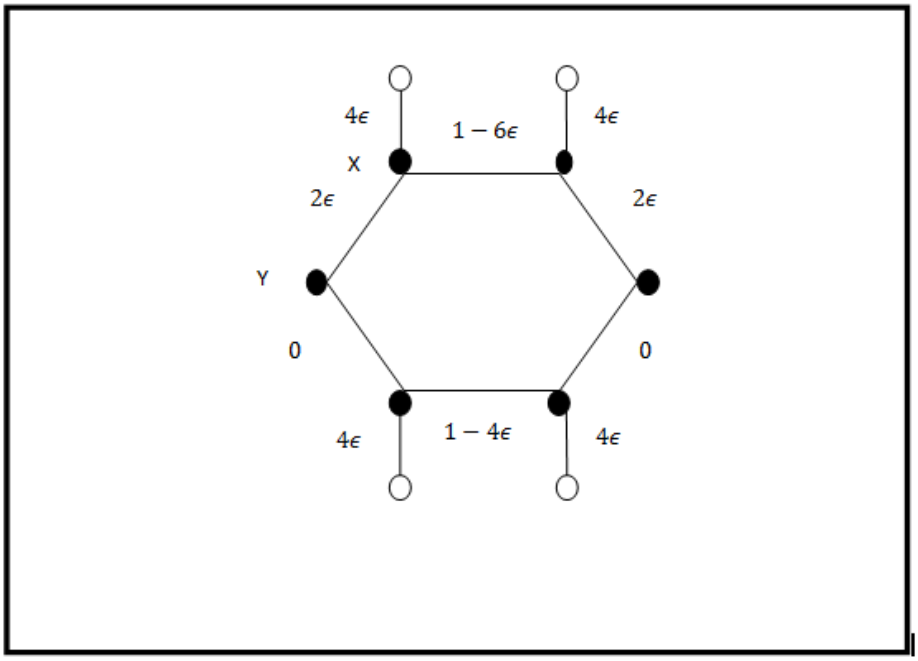


Figure 4.9: New charge solution $q + 2\varepsilon\bar{q}$

Now consider the optimal DRP_{mcp} Solution:2 in Figure 4.6. Because of the positivity constraints on the edge (x,y) again we have $\theta \leq 2\varepsilon$.

In this way the edges (x,y) and (y,z) alternatively enter to set J_3 and leave J_3 in each DRP_{opt} iteration. In first iteration the charge value increases by $2*\varepsilon$, in second iteration by $2*2\varepsilon$, in 3rd iteration $3*2\varepsilon$ and so on.
So after n iterations the value of the solution is:
$2*\varepsilon+4*\varepsilon+6*\varepsilon+8*\varepsilon+\ldots+2n*\varepsilon \cong 2$ (Optimal solution value)
which implies $2\varepsilon*(1+2+3+4+5+\ldots+n) \cong 2$ and so we have $\varepsilon*n(n+1) \cong 2$ implies $n \cong \frac{2}{\sqrt{\varepsilon}}$. That is the number of DRP_{opt} iteration $n \cong \frac{2}{\sqrt{\varepsilon}}$ where ε is a very small number. Hence the claim.

4.7 Analysis of the Primal-Dual Algorithm

The Algorithm 4.7 provides the outline of the *primal-dual algorithm* for the *maximum charge problem*. This *primal-dual algorithm* uses our proposed phased approach (Algorithm 4.6) to solve the DRP_{mcp} optimally. If the input graph instance is bipartite than the steps 1 and 10 are redundant.

Now we analyze the running time of the primal-dual algorithm for the *maximum charge problem*. We call the step of computing an optimal solution to a DRP_{mcp} instance as a DRP_{opt} iteration. Our goal is to bound the number of DRP_{opt} iterations. We complete our proof in two steps.

Recall that, given a feasible charge q an augmenting path is an alternating path of odd length along which one can increase the value of q. First we show that given a feasible

Algorithm 4.7 Primal-Dual algorithm for MCP

Require: A graph $G = (V, E)$ with arbitrary capacities defined on every vertex and every edge. Denote any feasible charge solution to G as q_g

Ensure: Optimal solution to the *MCP*

```
Construct the bipartite equivalent G_B of G as described in Section 3.4.3
Start with a feasible charge solution q = 0 in G_B
OPT = No
while OPT ≠ Yes do
    Construct the sets J_1, J_2, J_3 with respect to the current charge solution q
    Construct the current instance of DRP_mcp
    Compute the optimal solution q̄_opt to the DRP_mcp instance using Algorithm 4.6
    if Σ_{e∈E} q̄_{opt_e} = 0 then
        OPT = Yes
        Convert q to q^g
        return q^g
    else
        Compute θ given by the equations 4.5, 4.6, 4.7, 4.8
        Compute q* = q + θ q̄_opt
    end if
end while
```

charge solution q the length of the shortest augmenting paths with respect to q is nondecreasing. After each DRP_{opt} iteration we calculate θ and the charge solution is updated by a linear combination of the current charge solution, θ and the optimal solution to the current DRP_{mcp} instance. The charge solution is updated along the augmenting paths with respect to a feasible DRP_{mcp} solution $\bar{q}$. The augmenting paths are discovered and augmented during the DRP_{opt} iteration. So these are also the augmenting paths with respect to the charge solution q. Our goal here is to show that, between two successive DRP_{opt} iterations the length of the shortest augmenting paths with respect to q does not decrease.

Lemma 4.7.1. *Given a bipartite graph instance, the length of the shortest augmenting paths with respect to any feasible charge solution q is non-decreasing.*

Proof. Given a bipartite graph instance $G_B = (U \bigcup V, E)$ of *MCP*, a feasible charge solution q and the associated DRP_{mcp} instance let us consider the construction process of the HT

with respect to a feasible DRP_{mcp} solution $\bar{q}$ described in Section 4.5.3. In the HT the start level contains all the unsaturated vertices with respect to $\bar{q}$ in U, the last level contains at least one unsaturated vertex with respect to $\bar{q}$ in V, along any positive edge the value of $\bar{q}_e$ can be increased by 1 and along any negative edge the value of $\bar{q}_e$ can be decreased by 1. After each DRP_{opt} iteration θ is determined in order to update the charge solution q without violating feasibility in *MCP*. θ is actually determined by either a vertex $v \notin J_1$ that is closest to be in set J_1 or an edge $e \notin J_2$ that is closest to be in set J_2 or an edge $e \notin J_3$ that is closest to be in set J_3. For worst case scenario assume that θ is determined by an edge. After updating the charge solution q to q_1 by θ that specific edge will disappear from the HT during the next DRP_{opt} iteration. Since the updated charge solution q_1 only removes edges from the HT during the next DRP_{opt} iteration, the length of the shortest augmenting paths with respect to q_1 cannot decrease. Hence the claim. □

In the second step of the argument we show that when an edge e comes out of J_2 or J_3 the length of the shortest augmenting path containing edge e increases by 2. We know that given any feasible charge solution q the set $J_2 = \{e \mid q_e = c_e\}$ and $J_3 = \{e \mid q_e = 0\}$. Suppose that the edge e goes into J_3 after DRP_{opt} iteration i, this means that there is an augmenting path that involves e as a negative edge in DRP_{opt} iteration i. Let j be the DRP_{opt} iteration in which e goes out of J_3, this means e is used as a positive edge in iteration j.

Lemma 4.7.2. *When any edge e goes out of J_2 or J_3 the length of the shortest augmenting path containing edge e with respect to a feasible charge solution q increases by 2.*

Proof. Let e be the edge as stated in the claim. Let i, j be the DRP_{opt} iterations after which e goes into J_3 and out of J_3 respectively. Without loss of generality we can assume that $j = i + 1$. Note that there is a shortest augmenting path of length k that uses e as a negative edge after DRP_{opt} iteration i. Let after iteration j there be a shortest augmenting path of length $k' \geq k$ (by previous lemma) that uses e as a positive edge. If $k' > k$ then the

claim holds, else $k' = k$. If $k' = k$ let us consider the symmetric difference of these two augmenting paths. The symmetric difference has two augmenting paths with total length less than $2k$, which implies that after DRP_{opt} iteration j there exist an shortest augmenting path of length less than k violating the assumption that the length of the shortest augmenting path after DRP_{opt} iteration i was k. Hence the claim. Similarly, we can prove the claim for the case when an edge e enters the set J_2 and comes out. The lemma follows. □

Next we combine Lemma 4.7.1 and Lemma 4.7.2 and state the following theorem:

Theorem 4.7.1. *The running time of the primal-dual algorithm is* $O(|E|^2|V|^2)$.

Proof. Each augmentation of current charge solution either adds a vertex to set J_1 or adds(or removes) an edge to set J_2 or J_3. We can bound the number of iterations require for all the vertices to enter set J_1 by $O(|V|)$. We can bound the maximum number of times an edge can go out of J_2 or J_3 by $O(|V|)$. As there are $|E|$ edges so there can be at most $O(|E||V|)$ DRP_{opt} iterations for which edges can go out of J_2 or J_3. Therefore the maximum number of augmentation or DRP_{opt} iteration is bounded by $O(|E||V| + |V| + 2) \approx O(|E||V|)$.

Each DRP_{opt} iteration takes $O(|E||V|)$ time. The theorem follows. □

Chapter 5

Experiments and Results

5.1 Introduction

In this chapter we present an empirical evaluation of the algorithms discussed in Chapter 4. In Section 5.2 we discuss the process of generating the experimental data. In Section 5.3 we present the results.

We use Python 2.6 for the implementation of the algorithms. All the experiments presented in this Chapter were conducted on a 3.00 GHz Intel Pentium 4 processor with 1 GB of RAM in Ubuntu 8.10 environment. For details of the implementation please refer to the Appendix A.

5.2 Data Sets

We evaluate the algorithms on random bipartite sparse and dense graphs and also on some random general sparse and dense graphs. The procedure for constructing a random dense graph is described below:

- We specify the number of vertices $n = |V|$.
- For bipartite case we specify $|U| = |V| = n/2$. We consider the set U contains all even indexed vertices and the set V contains all odd indexed vertices.
- We specify $pr = 0.5$ as the probability of an edge being present in between any two vertices

- For every pair of vertices we generate a random number c in between 0 and 1. If $c \leq pr$ we put an edge between the pair of vertices and we generate a random integer number between 1 and 10 to specify the capacity of that edge.

- For each vertex we generate an integer number between 1 and 10 to specify the capacity of that vertex.

To construct a random sparse graph we specify $pr = 0.2$ as the probability of an edge being present between any two vertices.

5.3 Experimental Results

In this section we present the empirical data obtained by running experiments on the random bipartite dense and sparse graphs and on the random general dense and sparse graphs. Tables 5.1 and 5.2 represents the experimental data for random bipartite dense and sparse graph respectively. The first and second columns of the tables represent the number of vertices and number of edges respectively. The third and fourth columns respectively represent the number of DRP_{opt} iterations (See Section 4.7) using the *topological erase approach* required to obtain the optimal solution to the *MCP* and the average time (in sec) for a single DRP_{opt} iteration. The fifth and sixth columns respectively represent the number of DRP_{opt} iterations using the *blocking charge approach* required to obtain the optimal solution to the *MCP* and the average time (in sec) required for a single DRP_{opt} iteration. Figures 5.1 and 5.3 graphically compares the running time of the two different approaches on bipartite dense and sparse graphs and Figures 5.2 and 5.4 graphically compares the total running time of the primal-dual algorithm.

Tables 5.3 and 5.4 represent the experimental data for random general dense and sparse

Table 5.1: Experiments on random bipartite dense graphs

$\|V\|$	$\|E\|$	Topological erase approach		Blocking charge approach	
		DRP_{opt} iterations	Time (in sec)	DRP_{opt} iterations	Time (in sec)
50	308	144	0.0157	147	0.0197
100	1238	372	0.0769	390	0.0882
150	2810	1010	0.2353	1124	0.2323
200	5012	1530	0.4713	2111	0.4133
250	7828	2628	0.8608	3118	0.7226
300	11248	3295	0.9906	3851	0.7545
350	15379	4943	1.9257	6128	1.2013
400	20043	5217	1.9848	6006	1.4464
450	25468	8307	4.9822	9726	2.4826
500	31246	8697	5.1010	10198	2.6776
550	37898	9715	5.5616	11641	2.9094
600	44781	10391	6.8109	12267	3.3248
650	52804	13002	9.0014	14769	4.6168

graphs respectively. The first and the third columns of the tables are the number of vertices and the number of edges in the input general graph respectively whereas the second and fourth columns are the number of vertices and the number of edges in the bipartite equivalent. The fifth, sixth, seventh and eighth columns represent number of DRP_{opt} iterations required to obtain the optimal solution to the *MCP* and the average time (in sec) required for a single DRP_{opt} iteration for the *topological erase approach* and *blocking charge approach* respectively. Figure 5.5 and 5.7 graphically compare the running time of the two different approaches on the general dense and sparse graphs and Figures 5.6 and 5.8 graphically compares the running time of the primal-dual algorithm.

By observing both the tables and the plots we can say that although the *blocking charge approach* requires to be executed more number of times as compared to the *topological erase approach* to obtain an optimal DRP_{mcp} solution but for large problem size the *block-*

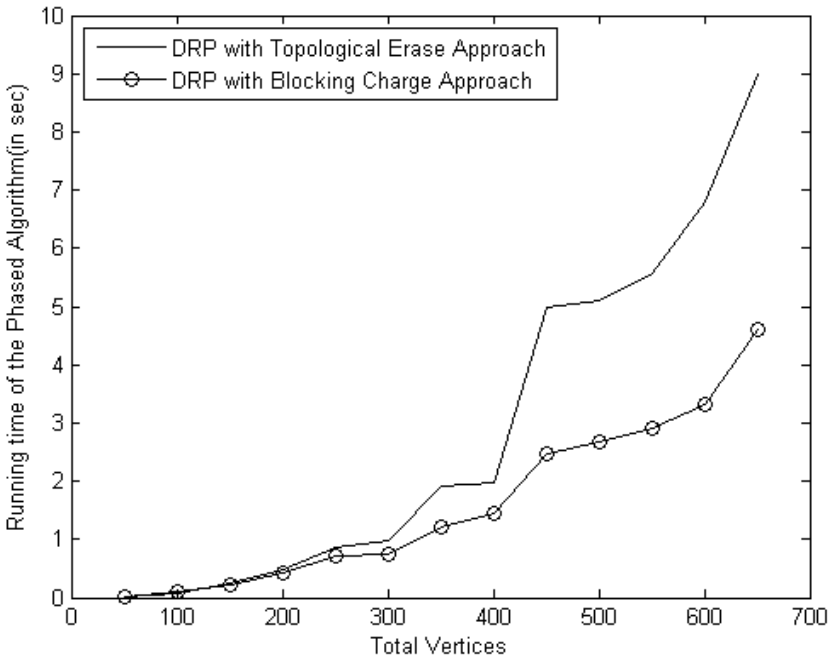


Figure 5.1: Comparison of the running time of the two different approaches on random bipartite dense graphs

Table 5.2: Experiments on random bipartite sparse graphs

$\|V\|$	$\|E\|$	Topological erase approach		Blocking charge approach	
		DRP_{opt} iterations	Time (in sec)	DRP_{opt} iterations	Time (in sec)
50	132	65	0.0081	65	0.0152
100	491	206	0.0385	206	0.0602
150	1078	540	0.1432	544	0.1714
200	1948	759	0.2113	829	0.3168
250	3101	1320	0.4007	1325	0.4526
300	4462	1882	0.6085	2004	0.7035
350	6171	2512	0.9563	3348	1.0726
400	7966	2715	1.2852	3659	1.3026
450	10178	2902	1.5791	3639	1.6585
500	12713	4503	2.9281	5734	2.4661
550	15042	5197	3.2745	6324	2.5999
600	17846	6490	3.9115	10486	3.4275
650	21131	7333	5.0187	10091	3.7812

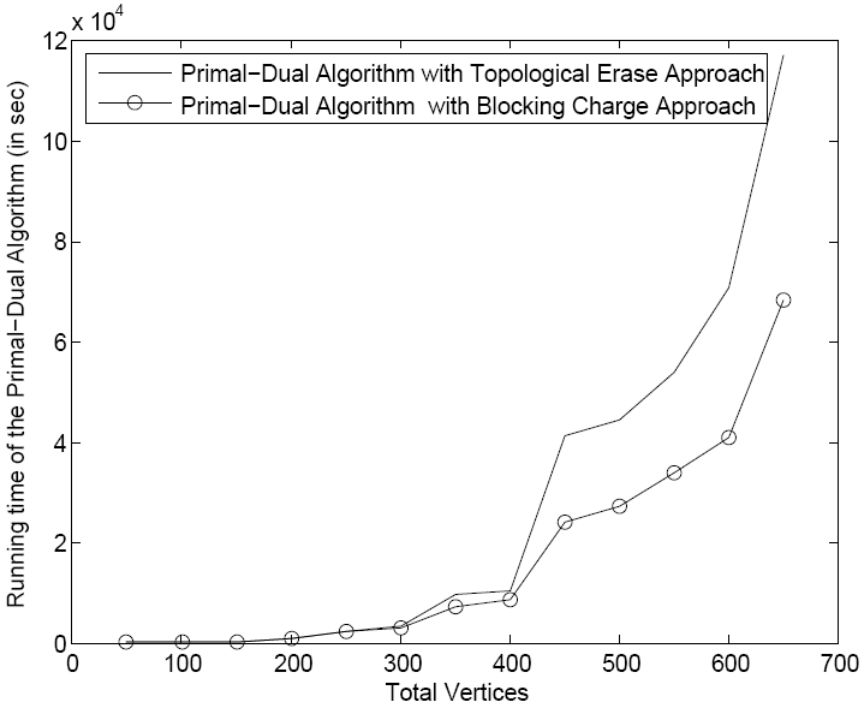


Figure 5.2: Comparison of the running time of the primal-dual algorithm with two different approaches on random bipartite dense graphs

Table 5.3: Experiments on random general dense graphs

$\|V\|$		$\|E\|$		Topological erase approach		Blocking charge approach	
General	Bipartite	General	Bipartite	DRP_{opt} iterations	Time (in sec)	DRP_{opt} iterations	Time (in sec)
50	100	600	1200	535	0.1022	543	0.1030
100	200	2404	4808	1998	0.4383	2105	0.3794
150	300	5589	11178	3908	1.4576	4951	1.0500
200	400	9954	19908	6821	2.8173	8557	1.7512
250	500	15527	31054	9117	6.2356	10425	2.8014
300	600	22399	44798	11668	7.2492	13916	3.6526
350	700	30491	60982	16723	15.4528	18652	5.5945
400	800	39855	79710	19056	17.5128	21120	7.4527

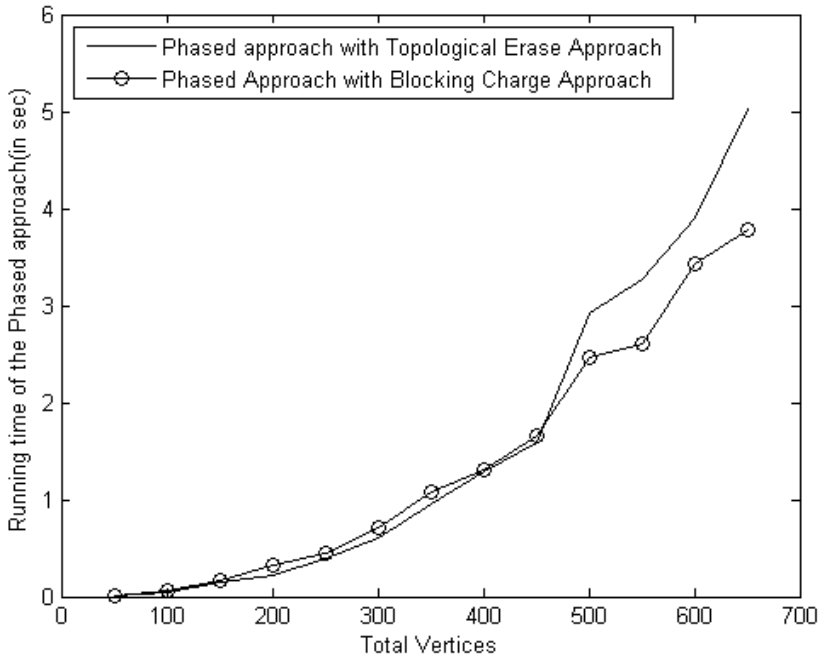


Figure 5.3: Comparison of the running time of the two different approaches on random bipartite sparse graphs

Table 5.4: Experiments on random general sparse graphs

$\|V\|$		$\|E\|$		Topological erase approach		Blocking charge approach	
General	Bipartite	General	Bipartite	DRP_{opt} iterations	Time (in sec)	DRP_{opt} iterations	Time (in sec)
50	100	232	464	244	0.0496	252	0.0776
100	200	947	1894	881	0.3074	957	0.3999
150	300	2311	4622	2193	0.9321	2777	0.9173
200	400	4007	8014	3148	1.5628	4105	1.3937
250	500	6296	12592	5253	2.3847	6665	2.0505
300	600	9029	18058	6933	5.0168	9039	3.5748
350	700	12301	24602	9256	6.5679	12709	4.5196
400	800	16041	32082	10277	7.6874	13395	5.6991

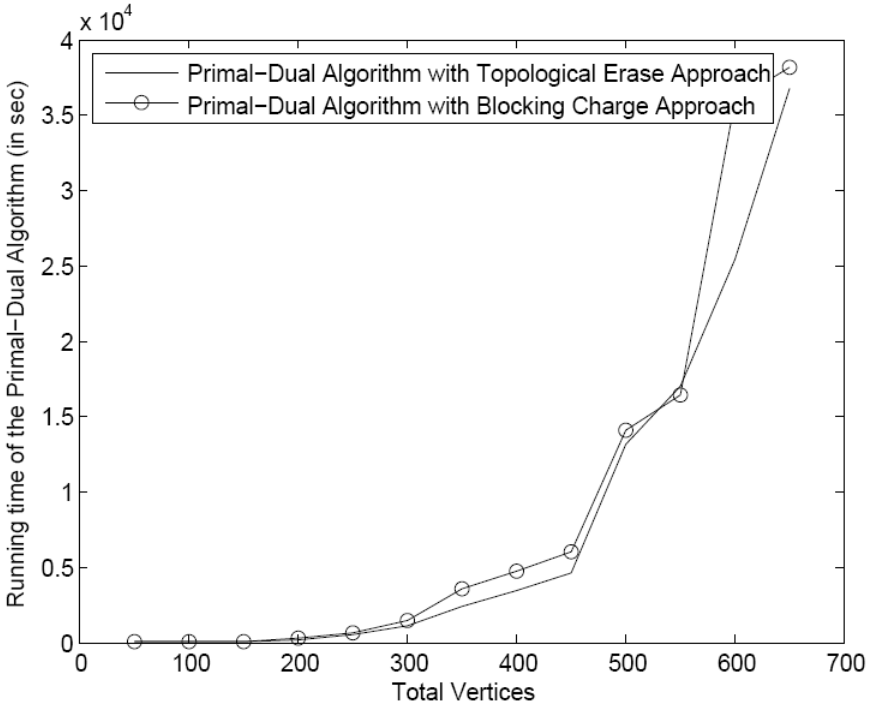


Figure 5.4: Comparison of the running time of the primal-dual algorithm with two different approaches on random bipartite sparse graphs

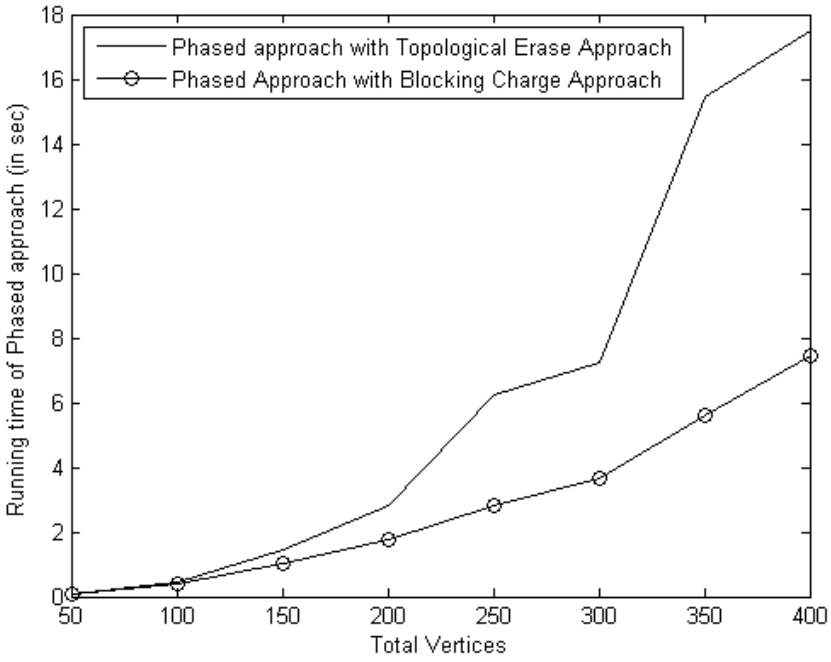


Figure 5.5: Comparison of the running time of the two different approaches on random general dense graphs

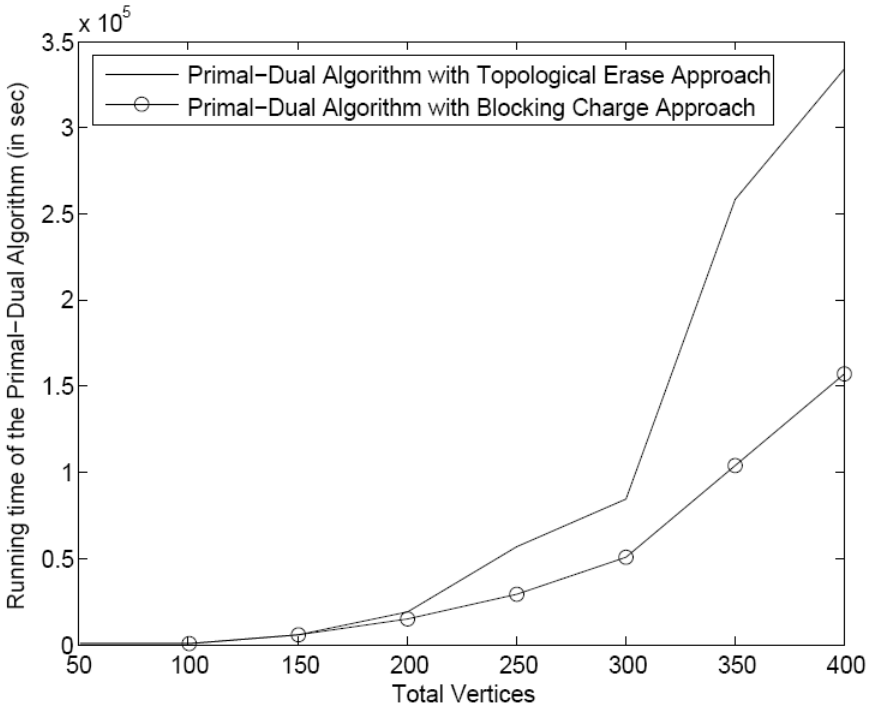


Figure 5.6: Comparison of the running time of the primal-dual algorithm with two different approaches on random general dense graphs

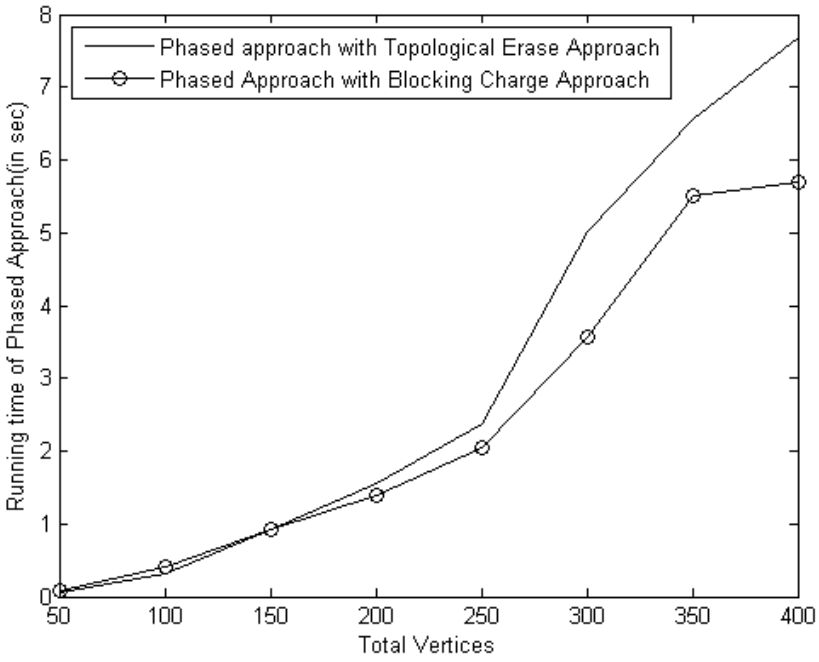


Figure 5.7: Comparison of the running time of the two different approaches on random general sparse graphs

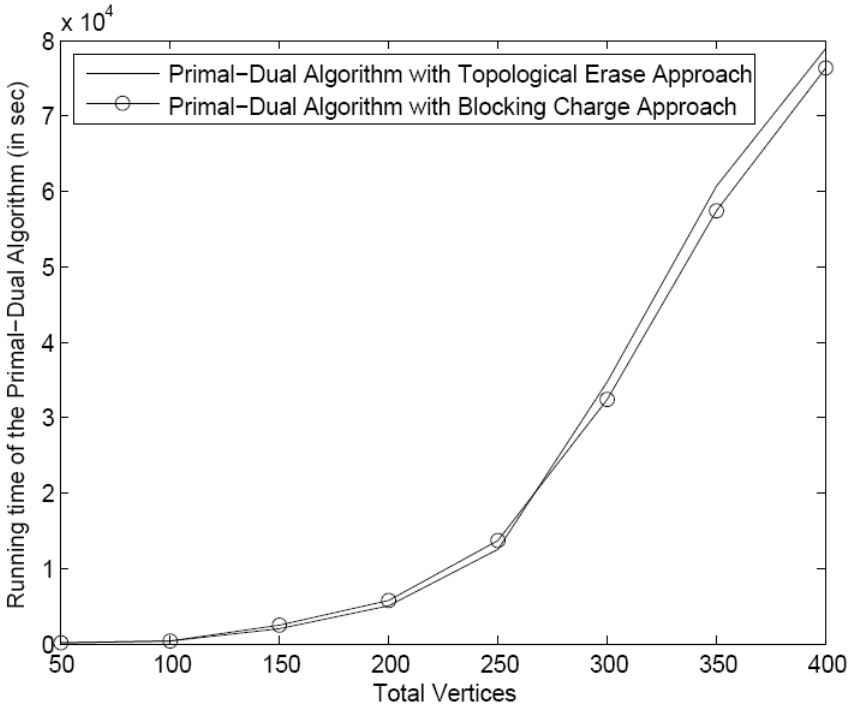


Figure 5.8: Comparison of the running time of the primal-dual algorithm with two different approaches on random general sparse graphs

ing charge approach computes an optimal solution to the DRP_{mcp} faster. All the plots show that for small graphs both the approaches have almost similar performance and sometimes the *topological erase approach* performs better. But as the size of the graph increases the *blocking charge approach* shows significant improvement in running time.

According to the tables the running time of the *topological erase approach* increases sharply with decrease in the vertex-edge ratio whereas for the *blocking charge approach* the increment is more steady. The reason behind is that as the vertex-edge ratio decreases so the number of augmenting paths increases and so the number of augmentations increases. As the *topological erase approach* takes one step for each augmentation so the time taken by each phase increases significantly and so the total time for approach to solve the DRP_{mcp} optimally. But in case of *blocking charge approach* the increment in running time is not so significant because sets of augmentations are done in parallel.

For the above reason the improvement in the running time of the primal-dual algorithm with blocking charge approach is more significant in case of the dense graphs as compared to the sparse graphs (see Figure 5.6, Figure 5.8). This improvement is achieved due to the technique of discovering and augmenting all the augmenting paths passing through a vertex in parallel instead of discovering and augmenting them one by one.

From the above discussion we can conclude that for large problem size the *primal-dual algorithm for maximum charge problem* using *blocking charge approach* shows better performance in terms of running time as compared to the one using *topological erase approach*.

Chapter 6

Conclusions

6.1 Summary

In this thesis we present a $O(|E|^2|V|^2)$ primal-dual algorithm for the *maximum charge problem* with capacity constraints. We characterize the optimal solution to the dual of the restricted primal DRP_{mcp} for the *maximum charge problem*. We design a *phased approach* for solving the DRP_{mcp} optimally.

In the first step of the phased approach we use the alternative breadth first search technique for constructing a Hungarian Tree with respect to the current DRP_{mcp} solution.

In the second step of the phased approach, we modify the *topological plus edge delete approach* due to Bobby Chan [6] in the unconstrained fractional matching problem and apply the modified approach for finding all the same length augmenting paths in the HT and augmenting them simultaneously. We call this approach as *topological erase approach*. We show that in worst case this approach can take $O(k|E|)$ time.

We then propose the *blocking charge approach* to discover all the same length augmenting paths in the HT and augment them in parallel. We show that this approach take $O(|E|)$ time. This is a improvement over the *topological erase approach* in the worst case. We also show that, using *blocking charge approach* the phased approach require $O(|E||V|)$ time to solve the DRP_{mcp} optimally.

We also show with example that choosing an augmenting path other than the shortest one

may cause the number of DRP_{opt} iterations to be unbounded.

We empirically show that for large problem sizes the *blocking charge approach* shows significant improvement in running time as compared to the *topological erase approach*.

6.2 Future Research

In future we plan to search for a technique to improve the running time complexity of the primal-dual algorithm for the *maximum charge problem*. It would be interesting to develop an $O(|E||V|)$ primal-dual algorithm for the problem.

Appendix A

A.1 Implementation

The basic data structures that we use for our implementation purpose are lists and list of lists.

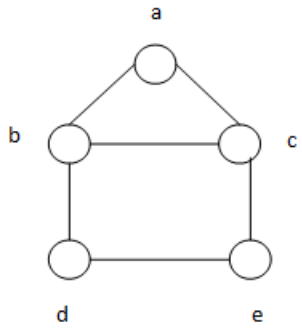
- For a bipartite graph $G_B = (U \bigcup V, E)$ we maintain three lists g_x, g_y and `NodeCapacity` to store the indices of the vertices in set U, the indices of the vertices in set V and the capacity on the vertices respectively. We also maintain the weighted adjacency list `graph`.

- For general graph $G = (V, E)$ we maintain the weighted adjacency list `InputGraph`. We maintain another list `NodeCapacityInputGraph` to store the capacities on the vertices.

- We maintain a list J_1 to store the index of the vertices in set J_1 (see Section 4.4). We also maintain two lists of list J_2 and J_3. The entries $J_2[i][j]$ and $J_2[j][i]$ contain 1 if the edge $e = (i, j)$ is in set J_2 (see Section 4.4) and similarly $J_3[i][j]$ and $J_3[j][i]$ contain 1 if the edge $e = (i, j)$ is in set J_3.

- For storing the associated `DRP` instance of the bipartite graph instance we maintain a list of lists `DRPInstance`.

- We maintain a list of the lists `level` where `level[i]` contains the index of the vertices in the i_{th} level of the `HT`. We also maintain a list of lists `dag` to store the adjacency list of the `HT`. We maintain a list `parent` that contains the current parent of the vertices in the `HT`.



$$\texttt{InputGraph} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \texttt{NodeCapacityInputGraph} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

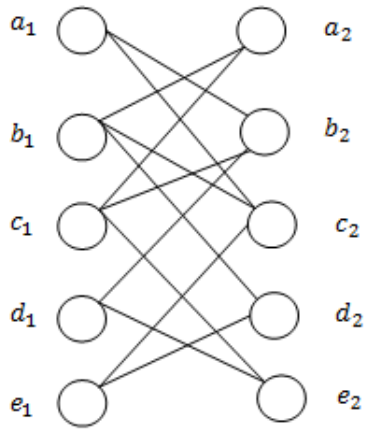
Figure A.1: A graph, corresponding weighted adjacency list and the list of node capacities

The Figure A.1 shows a graph and the weighted adjacency list `InputGraph` for the graph. For this graph the capacities on the edges and the vertices are considered as 1.
The Figure A.2 shows the bipartite equivalent of the graph in Figure A.1 and the weighted adjacency list `graph` for the bipartite graph.
The basic functions used in our implementation are:

- `BipartiteGraphReduction(InputGraph, NodeCapacityInputGraph, n)`: This function converts a general graph `InputGraph` to its bipartite equivalent using the technique described in Section 3.4.3 and Section 3.4.3.

- `Construct`J_1`(graph, ChargeSolution,` g_x`,` g_y`, NodeCapacity)`: This function checks for each vertex v if the sum of the charges on the edges incident on v is equal to the capacity of v. If so than v is appended to list J_1.

- `Construct`J_2J_3`(graph, ChargeSolution,` g_x`,` g_y`)`: This function checks whether



$$\texttt{graph} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \texttt{NodeCapacity} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure A.2: A bipartite graph, corresponding weighted adjacency list and the list of node capacity

the charge on each edge in the graph instance is equal to the capacity of the edge or zero . If the charge on an edge $e = (u, v)$ is equal to zero then the entries $J_3[u][v]$ and $J_3[v][u]$ are set to 1. If the charge on an edge $e = (u, v)$ is equal to its capacity then the entries $J_2[u][v]$ and $J_2[v][u]$ are set to 1.

- `CreateDrpInstance(graph, g_x, g_y, n, J_2, J_3)`: This function creates the DRP_{mcp} instance given the bipartite graph instance using the procedure describe in Section 4.5.1.

- `CalculateOptimalSolutionToDrp(DrpInstance, NodeCapacityDrpInstance, n, g_x, g_y, J_2, J_3)`: This function takes the lists `Drpinstance`, `NodeCapacityDrpInstance`, number of vertices n, the lists g_x, g_y, J_2, and J_3 as arguments. It implements the phased approach and returns optimal solution to the current *DRP* instance. This function calls the function `ConstructHT(`g_x`,`g_y`,DRPInstance,NodeCapacityDrpInstance, DrpSolution,`J_2`,`J_3`)` and one of the functions `TopologicalEraseApproach(dag,level, parent,NodeCapacityDrpInstance)` and `FindBlockingCharge(dag,drpsolution,level, lastlevel,n)`. These two functions are described in detail in Sections A.1.1 and A.1.2 respectively.

- `ConstructHT(`g_x`,`g_y`,DRPInstance,NodeCapacityDrpInstance,DrpSolution,`J_2`,`J_3`)`: This function constructs the HT using Algorithm 4.1 described in Section 4.5.3. It takes the lists g_x, g_y, the DRP instance `DRPinstance`, the lists `NodeCapacityDrpInstance`, J_2, J_3 and the list containing current DRP solution as arguments and returns the list of lists `level` containing the list of vertices at each level of the `HT`, the adjacency list of the HT `dag` and a list `parent` containing the current parent of the vertices in HT.

- `CalculateTheta(graph,NodeCapacity,ChargeSolution,DrpSolution,`J_1`,`J_2`,`g_x`,`g_y`)`: This function calculates θ using the equations 4.5, 4.6, 4.7 and 4.8.

- `FindMaxCharge(graph,`g_x`,`g_y`,NodeCapacity,n)`: This function implements the

primal-dual algorithm for the MCP as described in Algorithm 4.7. This function takes the weighted adjacency list `graph` of the input bipartite graph, the lists g_x, g_y, list containing the capacity of the vertices `NodeCapacity` and number of vertices n as arguments and returns the optimal solution to the maximum charge problem. It starts with a `ChargeSolution` containing 0 on all the edges and initializes the sets J_1, J_2 and J_3. It then calls the functions `CreateDrpInstance(graph,`g_x`,`g_y`,n,`J_2`,`J_3`)` to obtain the current *DRP* instance. It then calls the function `CalculateOptimalSolutionToDrp(DrpInstance, NodeCapacityDrpInstance,n,`g_x`,`g_y`,`J_2`,`J_3`)` to obtain an optimal solution to the current DRP instance. It then calls the function `CalculateTheta(graph,NodeCapacity, ChargeSolution,DrpSolution,`J_1`,`J_2`,`g_x`,`g_y`)` to obtain the θ value and using the linear combination of the current `ChargeSolution`, θ and `DrpSolution` calculates the improved charge solution. It then calls the functions `Construct`J_1`(graph,Chargesolution,`g_x`,`g_y`, NodeCapacity)` and `Construct`J_2J_3`(graph,ChargeSolution,`g_x`,`g_y`)` to update the sets J_1, J_2 and J_3 with respect to the current *ChargeSolution*. It then updates the current `DrpInstance` using the new J_1, J_2, J_3. The whole process is repeated until a `drpsolution` with optimal objective function value 0 is obtained.

A.1.1 Implementation of the Topological Erase Approach

In this section we discuss the implementation procedure of the *topological erase approach* for finding and augmenting the augmenting paths in a HT.

The function `TopologicalEraseApproach(dag,level,parent,NodeCapacityDrpInstance, DrpSolution)` takes the lists `dag`, `level`, `parent`, `NodeCapacityDrpInstance`, `DrpSolution` as arguments, checks for the augmenting paths with respect to current DRP solution `DrpSolution`

in the dag using the lists level and parent and augments the current DrpSolution along the augmenting paths. It uses the procedure described in Algorithm 4.2 and returns the updated DrpSolution. It calls the functions PathFinder(node,level,parent) to compute an augmenting path from a vertex in the last level to the first level of the HT, UpdateDrpSolution(DrpSolution,path) to augment the current DRP solution along an augmenting path, DeleteSaturatedEdges(dag,DrpSolution,DrpInstance,J_2, J_3,parent,level,path) to delete the saturated edges along an augmenting path and DeleteVertexWithNoIncomingEdges(path,dag,parent,level) to delete a vertex with no incoming edges from the HT.

A.1.2 Implementation of the Blocking Charge Approach

In this section we discuss the implementation procedure of the *blocking charge approach* for finding and augmenting the augmenting paths in a HT.

The function FindBlockingCharge(dag,drpsolution,level,lastlevel,n) takes the weighted adjacency list dag, the current DRP solution drpsolution, the list level, the index of last level of HT lastlevel, number of vertices n as arguments and uses the procedure described in Algorithm 4.3 to compute the blocking charge in the current HT. This function calls the functions CalculateInAndOutCapacityOfVertices(n, level, lastlevel, dag) to calculate total incoming capacity and total outgoing capacity of all the vertices in the level; CalculateDeficitForVertices(n, level, lastlevel, InCapacityOfVertices, OutCapacityOfVertices) to compute the minimum deficit vertex; PushChargeDrpUptoLastLevel(dag, mindeficit,mindeficitvertex, LevelOfMinDeficitVertex, level, lastlevel, DrpSolution, DeficitOfVertices, InDrpSolutionOfVertex, OutDrpSolutionOfVertex, InCapacityOfVertices,

`OutCapacityOfVertices)` to transfer the `mindeficit` amount of charge to the last level using Procedure 4.4;
`PullChargeDrpFromFirstLevel(dag, mindeficit, mindeficitvertex, LevelOfMinDeficitVertex, level, lastlevel, DrpSolution,DeficitOfVertices, InDrpSolutionOfVertex, OutDrpSolutionOfVertex, InCapacityOfVertices, OutCapacityOfVertices)` to transfer the `mindeficit` amount of charge from the first level to the `LevelOfMinDeficitVertex` using Procedure 4.5 and `updatedag(dag, DeficitOfVertices, mindeficitvertex, LevelOfMinDeficitVertex, level, lastlevel, drpsolution, IncapacityOfVertices, OutCapacityOfVertices)` to delete all the incident edges of the `mindeficitvertex` and to update the incapacity and outcapacity of the neighboring vertices when minimum deficit is zero.

Bibliography

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flow: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] M. H. Alsuwaiyel. *Algorithms Design Techniques and Analysis*. World Scientific Publishing, 1999.

[3] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43:842–844, 1957.

[4] R. G. Bland. New finite pivoting rules. Discussion Paper 7612, Center for Operation Research and Econometrics, Universite Catholique de Louvain, Heverlee, Belgium, June, 1976.

[5] J. M. Bourjolly and W. R. Pulleyblank. König-egerváry graphs, 2-bicritical graphs and fractional matchings. *Discrete Applied Mathematics*, 24:63–82, 1989.

[6] Bobby Chan. A primal-dual algorithm for the unconstrained fractional matching problem. Master's thesis, Simon Fraser University, 2009.

[7] V. Chávtal. *Linear Programming and Extensions*. W.H. Freeman and Company, 1983.

[8] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[9] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 1965.

[10] J. Hopcroft and R. Karp. An $O(n^{5/2})$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, 2(4):225–231, 1973.

[11] T. Kameda and I. Munro. An $O(|V||E|)$ algorithm for maximum matching of graphs. *Computing*, 12(1):91–98, March 1974.

[12] Narendra Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[13] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[14] L. G. Khachiyan. A polynomial time algorithm for linear programming. *Soviet Mathematics Doklady*, 20:1092–1096, 1979.

[15] V. Klee and G. J. Minty. How good is the simplex algorithm ? *Inequalities III*, pages 159–172, 1972.

[16] D. Kozen. *The Design and Analysis of Algorithms*. Springer, 1992.

[17] R. Krishnamurti, D. R. Gaur, S. K. Ghosh, and H. Sachs. Berge's theorem for maximum charge problem. *Discrete Optimization*, 3:174–178, 2006.

[18] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[19] L. Lovász and M. Plummer. *Matching Theory*. North-Holland, 1986.

[20] Vishv M. Malhotra, M. Pramodh Kumar, and S. N. Maheshwari. An $O(|V|^3)$ algorithm for finding maximum flows in networks. *Information Processing Letters*, 7(1):277–278, January 1978.

[21] M. Mucha and Piotr Sankowski. Maximum matching via gaussian elimination. In *Proceedings. 45th Annual Symposium on Foundations of Computer Science*, pages 248–255, 2004.

[22] C. H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc., 1982.

[23] R. Rizzi. A short proof of König's matching theorem. *Journal of Graph Theory*, 2000.

[24] Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory A Rational Approach to the Theory of Graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization, 1997.

[25] Robert J. Vanderbei. *Linear Programming : Foundations and Extensions*. Princeton University, 2 edition, 2001.

[26] V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{|V||E|})$ general graph matching algorithm. *Combinatorica*, 14(1):71–109, 1994.