ON PRIMAL-DUAL SCHEMA FOR THE MINIMUM SATISFIABILITY PROBLEM

UMAIR MUHAMMAD ARIF
Bachelor of Engineering (Electronic)
NED University of Engineering and Technology, 2011

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

ON PRIMAL-DUAL SCHEMA FOR THE MINIMUM SATISFIABILITY PROBLEM

UMAIR MUHAMMAD ARIF

Date of Defense: December 22, 2016

Dr. Daya Gaur Co-supervisor	Professor	Ph.D.
Dr. Robert Benkoczi Co-supervisor	Associate Professor	Ph.D.
Dr. Shahadat Hossain Committee Member	Professor	Ph.D.
Dr. Saurya Das Committee Member	Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

I dedicate this thesis to my **parents** especially my **mother** whose uncountable prayers made this possible.

Abstract

Satisfiability problem is the first problem known to be NP-complete [8, 28]. In this thesis, we have studied the minimization version of the satisfiability problem called the MINSAT. Given a set of boolean variables and a set of clauses, such that each clause is a disjunction of variables, the goal is to find the boolean values of the variables so that minimum number of clauses are satisfied. We have used the concept of linear programming and the primal-dual method to study the problem. We have constructed the Linear program of the MINSAT and its restricted version. We have proposed two combinatorial methods to solve the dual of the restricted primal of the MINSAT. Further to this, these two algorithms also obtain an integral solution to the dual of the MINSAT problem. Lastly, we performed a comparison analysis of our proposed algorithms with the simplex method.

Acknowledgments

I would like to start my aknowledgement with two of the best people I found in Canada. They are my supervisors Dr. Daya Gaur and Dr. Robert Benkoczi. Both of them were like mentor to me. Their motivation, guidance, appreciation and time devotion at each and every stage of my graduate studies was unforgettable. Without these two personalities, this day would never be possible.

In addition to them, I would like to deeply thank my committee members Dr. Shahadat Hossain and Dr. Saurya Das. They were always there whenever I need any precious advice at every stage of this journey. Their comments and feedback were highly helpful. A special thanks to Dr. Ramesh Krishnamurty for his valuable suggestions. I would also like to extend my thanks to Dr. Howard Cheng for being the Examination chair of my thesis defense. I am highly grateful to my supervisors and SGS for the financial assistantships.

Last but not the least, a token of appreciation and thanks to all the people associated with the Maths and Computer Science department. Be it the faculty members, fellow graduate students or the ever-helpful Admin Assistant Barbara Hodgson, they all made my stay at this department worthwhile. All the graduate students were close to me and I would like to thank all of them. Some of the notable names (in random order) are Nabi, Sara, Kawsar, Fariha , Arnab, Jayati, Ram, Mark, Anik, Marzia, Suny, Mithila, Parijat, Nurgul, Samanta, Tafseer, Imtiaz, Lazima, Sharmin and Rafat. I am grateful to you guys for being a part of my life at the University of Lethbridge.

Contents

Contents vi

List of Tables viii

List of Figures ix

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Organization of the thesis . . . 2

2 Preliminaries and Related Research 3
- 2.1 Introduction . . . 3
- 2.2 Definitions of Important Terms . . . 3
  - 2.2.1 Graph . . . 3
  - 2.2.2 Decision Problems . . . 3
  - 2.2.3 Optimization Problems . . . 4
  - 2.2.4 Class P . . . 4
  - 2.2.5 Class NP . . . 4
  - 2.2.6 NP-complete and NP-hard . . . 5
  - 2.2.7 Approximation Algorithms . . . 5
- 2.3 Linear Programming and Duality Theory . . . 6
- 2.4 Methods for solving Linear Programs . . . 9
- 2.5 The Simplex method . . . 10
  - 2.5.1 Simplex Algorithm in General form . . . 13
  - 2.5.2 Choosing the initial basic feasible solution . . . 14
- 2.6 Primal Dual Method . . . 15
  - 2.6.1 Duality Theory . . . 16
  - 2.6.2 Complementary Slackness . . . 18
  - 2.6.3 Primal-Dual Schema . . . 19

3 The Minimum Satisfiability problem 25
- 3.1 Introduction . . . 25
- 3.2 The Satisfiability Problem . . . 25
  - 3.2.1 Definition . . . 25
  - 3.2.2 Problem Definition . . . 26
- 3.3 The MINSAT Problem - Definition . . . 27
- 3.4 Previous Research Work . . . 27
- 3.5 MINSAT as an Integer Linear Program . . . 30

3.5.1 Primal Formulation . . . 31
3.5.2 Dual Formulation . . . 31
3.6 Primal-dual Schema for MINSAT . . . 32
3.6.1 Restricted Primal . . . 33
3.6.2 Dual of Restricted Primal . . . 33
3.7 Our Contributions . . . 35

4 Combinatorial Algorithms for the DRP, an Exploration 36
4.1 Introduction . . . 36
4.2 Algorithm 1: Finding a feasible path in a bipartite graph . . . 36
4.2.1 Step 1: Formulation of DRP and construction of bipartite graph . . . 38
4.2.2 Step 2: Finding a forward feasible path and DRP solution . . . 39
4.2.3 Step 3: Finding the final DRP and Dual solution . . . 43
4.3 Algorithm 2: Edge picking in a graph . . . 45
4.3.1 Step 1: Formulation of DRP and construction of bipartite graph . . . 46
4.3.2 Step 2: Picking edges to solve DRP . . . 46
4.4 Our Contributions . . . 48

5 Experiments and Results 50
5.1 Introduction . . . 50
5.2 MINSAT Instances . . . 50
5.3 Empirical Evaluation . . . 51

6 Conclusion 58
6.1 Summary . . . 58
6.2 Future Research . . . 58

Bibliography 59

List of Tables

5.1 Experimental results on number of iterations for the three algorithms 53
5.2 Experimental results on the processing times of the two proposed algorithms 55
5.3 Experimental results on quality of solution (Approximation Ratio) for the two algorithms 56

List of Figures

2.1 Flowchart for Primal Dual Schema 20

4.1 An example of a graph having 8 vertices and 8 edges 38
4.2 An example of a bipartite graph 39

5.1 A comparison of the number of iterations of the two algorithms with the Simplex method 54
5.2 A comparison of the number of iterations between the two proposed algorithms 54
5.3 A comparison of the processing times of the two proposed algorithms . . . 55

Chapter 1

Introduction

1.1 Introduction

The field of Computer science is incomplete without the boolean values 0 and 1. These two values were responsible for the introduction of logic theory and satisfiability. The satisfiability problem aims at finding the boolean values of a set of variables such that these values satisfy a given formula. We will formally define the satisfiability problem in Chapter 3. The satisfiability problem finds its applications in a number of areas which include but not limited to artificial intelligence, formal verification, electronic design automation, etc. A book by Biere et al [6] is fully dedicated on this problem. The book contains the history, complexities, variants, practical solving and applications of the satisfiability problem.

There are two optimization versions of the satisfiability problem. We have considered the minimization version of the satisfiability problem in this thesis. The problem is called the minimum satisfiability problem or MINSAT. The satisfiability problem is the first known NP-complete [8, 28] problem. By the definition of NP-completeness [25], any problem in NP can be reduced to the satisfiability problem in polynomial time. The process is called SAT encoding. This serves as the motivation to design fast and efficient SAT solvers.

We have modelled the MINSAT as an Integer linear program (ILP) and applied primal-dual method to investigate the existence of an algorithm that solves the problem combinatorially instead of using the numerical simplex method. We have proposed two such algorithms to solve the dual of the restricted version of the MINSAT LP. Although, our

algorithms did not guarantee an optimal solution but gave an integral solution of the dual LP unlike the fractional solution given by the simplex method.

1.2 Organization of the thesis

We start by defining the terminologies that will be used in this thesis in Chapter 2. We also define the fundamentals of linear programming and the methods used to solve a linear program in this chapter. We describe in detail two of such methods.

In Chapter 3, we formally define the satisfiability problem and the minimum satisfiability problem. We also discuss the previous research work in the area. Then, we construct the related linear programs for the MINSAT problem in the context of the primal-dual method.

In Chapter 4, we present the two approaches that we have investigated to solve the Dual of the Restricted Primal (DRP) of the MINSAT combinatorially. We analyze both the algorithms with reference to their implementation and running time complexities.

In Chapter 5, we give the empirical results obtained from the implementation of the proposed algorithms. We also give a comparative analysis of the proposed algorithms with the simplex method with the help of experimental results.

Finally, we conclude the thesis in Chapter 6 with listing the future research possibilities and directions.

Chapter 2

Preliminaries and Related Concepts

2.1 Introduction

In this chapter, we will discuss the terminology and concepts used in this thesis. We will start by discussing the definitions that will be used throughout this document in section 2.2. Later, in section 2.3, we will go through the basics of linear programming. We will describe some methods for solving the linear programming problems in section 2.4. Then, we will discuss in detail two of these methods in sections 2.5 and 2.6.

2.2 Definitions of Important Terms

These definitions are from the books by Cormen [9], Kleinberg and Tardos [25], Vazirani [38] and Williamson and Shmoys [37].

2.2.1 Graph

A graph G is a pair of sets (V,E). V is a set of nodes (also called *vertices*) while E is a set of edges, each of which joins two of the nodes. The edges can be directed from one vertex to another, if such is the case then G is a directed graph. Similarly, each edge can have a weight (usually a non-zero number) which makes G a weighted graph.

2.2.2 Decision Problems

A given problem is called a decision problem if it has a single possible answer which can be either *yes* or *no*.

Example: Given an undirected unweighted graph $G = (V, E)$, two vertices $u, v \in V$ and a non-negative integer k. Is there a path [25] in G between u and v whose length is at most k? So the answer to this problem is simply *yes* or *no*.

2.2.3 Optimization Problems

A given problem that asks us to find a solution out of numerous feasible solutions that either maximizes or minimizes a given objective.

Example: Given an undirected unweighted graph $G = (V, E)$ and two vertices $u, v \in V$, find the shortest path between u and v. So the answer to this problem can be more than one, i.e., there can be many paths between these two vertices but we have to find the shortest one out of those available paths.

2.2.4 Class P

The problems that are solvable in polynomial time on a deterministic Turing machine fall under this category. The term polynomial time refers to any algorithm that runs in time $O(n^k)$ for some constant value of k where n is the size of the input.

Finding the shortest path between two vertices of a given graph is in class P.

2.2.5 Class NP

The decision problems which can be verified in polynomial time on a deterministic Turing machine. For NP problems, we are given a certificate of the correctness of the solution which can be verified in polynomial time of the input size. The term NP stands for Non-deterministic polynomial time.

Example: Consider the vertex cover problem in which we are given a graph $G = (V, E)$ and an integer k. We are asked whether there is any subset $C \subseteq V$ such that every edge $(u, v) \in E$ has at least one endpoint (u or v) in C and $|C| = k$? So, if someone claims that C is a vertex cover, then we can easily check in polynomial time whether $|C| = k$ and for every edge, at least one of the endpoints is in set C or not.

2.2.6 NP-complete and NP-hard

First, we will define the term NP-complete. A problem X is said to be NP-complete, if the following two conditions hold:

1. The problem belongs to the class NP i.e. $X \in NP$.

2. Every other NP problem Y is polynomially reducible to X i.e. $Y \leq_p X$. Polynomial reduction means every instance of the problem Y can be transformed to an instance of X in polynomial time [25]. The transformation is such that answer of every instance of Y is "yes" if and only if the answer of the mapped instance of X is "yes'.

If a problem satisfies the second condition and may not satisfy the first condition, then it is called an NP-hard problem [9]. Till the writing of this thesis, no polynomial time algorithm has been discovered to solve any NP-complete problem nor anyone proved that such an algorithm doesn't exist. This fact brings our attention to a middle approach. The approach is to design efficient approximation algorithms for NP-hard optimization problems.

2.2.7 Approximation Algorithms

Williamson and Shmoys [38] have referred to an old engineering slogan "Fast, Cheap, Reliable, Choose two" to describe the term approximation algorithms. Whenever we are dealing with an NP-hard optimization problem, "Fast and Cheap" implies an efficient algorithm, while "Reliable" means it should give an optimal solution for every instance of input. It is generally believed that we do not have polynomial time algorithms for NP-complete problems so we have to compromise on one of these requirements which give birth to the field of approximation algorithms.

One can relax any one of the three factors. The most common approach is to relax the requirement of obtaining an optimal solution. We design polynomial time efficient algorithms to obtain a "good enough" solution. Formally, *an* α*-approximation algorithm is a polynomial time algorithm designed to solve any instance of an optimization problem such that the cost of its solution is bound within a multiplicative factor of* α *of the value of*

optimal solution. Where α is called the performance ratio and its value is:

$$\alpha = \frac{A(I)}{OPT(I)} \geq 1 \qquad \text{(for a minimization problem)}$$

$$\alpha = \frac{A(I)}{OPT(I)} \leq 1 \qquad \text{(for a maximization problem)}$$

where $A(I)$ is the cost of the solution returned by approximation algorithm A on an input instance I, and $OPT(I)$ is the optimal cost.

2.3 Linear Programming

Linear programming is a very powerful technique for solving optimization problems. This technique was first introduced by the Nobel-laureate Leonid Vitaliyevich Kantorovich in 1939 [32]. It is considered to be a vital tool for solving real-life operations research and planning problems. Due to its universality in modeling different optimization problems, it is also used as a tool to solve combinatorial optimization problems in the field of computer science.

The purpose of a linear program (LP) is to maximize or minimize a given linear function based on real-valued decision variables while satisfying a set or collection of linear constraints. This function is termed as the objective function. A generalized form [36] of this function looks like:

$$\zeta = c_1x_1 + c_2x_2 + ... + c_nx_n \tag{2.1}$$

ζ is the objective function value. $x_1, x_2, ..., x_n$ and $c_1, c_2, ..., c_n$ are the decision variables and their constant cost coeficients respectively. This optimization is done with some restrictions on the values of these decision variables. These restrictions are referred to as the *constraints*. The constraints are in the form of either inequalities or equalities. The constraints are broadly defined into two sub-categories shown in 2.2 (technological constraints) and 2.3 (non-negativity constraints).

$$a_i x_1 + a_i x_2 + ... + a_i x_n \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i \qquad where \quad i = 1,2,...,m \tag{2.2}$$

$$x_j \geq 0 \qquad where \quad j = 1,2,...,n \tag{2.3}$$

An LP is a combination of an objective function that needs to be optimized with the restriction that the given constraints must be satisfied. The number of decision variables is usually represented by n while number of constraints, excluding the non-negativity constraints, is denoted by m. A general LP formulation [7] for a maximization and minimization problem is shown in 2.4 and 2.5 respectively. It may be noted that when the objective function is of type maximization then the inequality constraints are in the form of a less than equal to inequality. On the other hand, for a minimization problem, the inequality constraint is in the form of a greater than equals inequality. This type of formulation is also called canonical form [34] of an LP. The canonical form is different from the standard form in which the inequality constraints are replaced with equality constraints. As we will discuss in later sections, a canonical form can easily be converted into a standard form by using slack or surplus variables [36].

$$maximize \quad \sum_{j=1}^{n} c_j x_j \tag{2.4}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1,2,...,m$$

$$x_j \geq 0 \qquad j = 1,2,...,n$$

$$\begin{aligned} \textit{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad i = 1,2,...,m \\ & x_j \geq 0 \qquad j = 1,2,...,n \end{aligned} \tag{2.5}$$

The general form of an LP can also be written using a matrix representation. We have shown the matrix formulation of both the LPs, maximization and minimization, in standard form in equations 2.6 and 2.7.

$$\begin{aligned} \textit{maximize} \quad & cx \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.6}$$

$$\begin{aligned} \textit{minimize} \quad & cx \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.7}$$

cx is the objective function. c is a row vector of costs and x is a column vector of decision variables. A is an m x n matrix called the coefficient matrix of the constraints while b is a column vector that represents the right-hand side of the constraints. A *solution* is an assignment of values to the variables x. A $feasible\ solution$ to an LP is an assignment of real values to the variables that satisfy all the constraints. If the objective function value obtained is maximum possible, then the feasible solution is called the $optimal\ solution$ for a maximization problem. Alternatively, if the value is the minimum of all the possible values, then it will be termed as an optimal solution for a minimization problem. Furthermore, if an assignment does not satisfy all the constraints, the solution is called an $infeasible\ solution$.

A linear program with additional constraints that the decision variables must take integer values is called an *integer linear program* (abbreviated as ILP). Many combinatorial problems can be expressed naturally as an ILP and usually serve as a first step to solve that problem using polynomial time approximation algorithm. We remove the integrality constraints on the variables and solve the relaxed LP. If an optimal solution is fractional for the relaxed version, we perform a rounding to get an approximate and integral solution to the ILP [35].

2.4 Methods for solving Linear Programs

There are numerous methods available to solve a linear program. The first and the most commonly used algorithm that was developed by George B. Dantzig [10] is the Simplex method. This method was based on finding the optimal solution iteratively by searching the corner points of the constraint polytope termed as the feasible region. Another method proposed by Kuhn [27] for solving the assignment problem which is a special LP is called the primal-dual method. The primal-dual method works for general LPs as well.

Other methods include the first polynomial time algorithm due to Khachiyan [24] and the interior-point method by Karmarkar [23]. The method proposed by Khachiyan was derived from the ellipsoid method developed by Shor, Yudi, and Nemirovskii [1]. The main idea behind the ellipsoid algorithm is to enclose the feasible solutions in an ellipse and then check iteratively whether the center of the ellipse is feasible or not. This algorithm lacks efficieny in practical scenarios compared to the simplex method. The interior point method starts with an interior point of the polytope and then repeatedly moves inside this polytope along some path to find an optimal solution. The interior point methods are practical in comparison to the ellipsoid method. For this thesis, we will only discuss the simplex method and the primal-dual method in detail.

2.5 The Simplex method

The simplex method is one of the most widely used iterative methods for solving a linear program. It was designed by George B. Dantzig [10] in the 1940s. The simplex method starts with an LP in standard form. As discussed previously, the standard form requires all the constraints, except for the non-negativity constraints, to be in equality form. If any constraint is an inequality, we add (or subtract) a slack variable. This slack (called surplus if subtracted) variable w will also appear in the objective function but with a coefficient $c_w = 0$.

Next example shows how we convert a given LP to standard form by introducing slack variables. The LP in 2.8 is in canonical form while the LP in 2.9 is in standard form. It can be seen that we have introduced slack variables w_1, w_2, w_3 and w_4. The number of slack variables introduced is at most the total number of constraint equations (referred as m). In reference to simplex method, the LP in the form of equation 2.9 is called a dictionary. The non-zero variables appearing on the left-hand side of the constraint equalities are called *basic variables* while the ones on the right-hand side are called *non -basic variables*. In other words, the basic variables are expressed as linear combination of the non-basic variables. Any solution $x_1, x_2, ..., x_n, w_1, w_2, ..., w_m$ obtained when all the non-basic variables are set to zero is called a *basic feasible solution*.

$$\begin{aligned} \textit{maximize} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & 2x_1 + x_2 \leq 4 \\ & 2x_1 + 3x_2 \leq 3 \\ & 4x_1 + x_2 \leq 5 \\ & x_1 + 5x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{2.8}$$

$$\begin{aligned} \textit{maximize} \quad & \underline{Z = 2x_1 + x_2} \\ \text{subject to} \quad & w_1 = 4 - 2x_1 - x_2 \\ & w_2 = 3 - 2x_1 - 3x_2 \\ & w_3 = 5 - 4x_1 - x_2 \\ & w_4 = 1 - x_1 - 5x_2 \\ & x_1, x_2, w_1, w_2, w_3, w_4 \geq 0 \end{aligned} \tag{2.9}$$

The simplex method is an iterative method that starts from an initial basic feasible solution and then moves to a better solution that improves our objective function. Here, the better value of the objective function for a minimization problem simply means a value smaller than the one previously obtained. While for a maximization problem, the improved value will usually be a value greater than its previous counterpart. We will illustrate the simplex method on the example in 2.9 and then describe the algorithm in general terms.

To start with, we need an initial basic feasible solution $(x_1, x_2, w_1, w_2, w_3, w_4)$. If the right-hand side of all the constraints is non-negative then the initial basic feasible solution can be found by setting all the non-basic variables to zero. Thus, the solution is (0,0,4,3,5,1). The value of objective function here is $Z = 0$.

The next step is to analyze how this objective function value can be increased. We examine the coefficients in the objective function and pick the one with the largest non-negative coefficient, other strategies also exist. In our example, we pick x_1 as any value greater than zero will increase our objective function by a factor of 2. We have to ensure the new value of x_1 does not violate non-negativity constraint for any variable. To achieve this, we find the maximum permissible value of x_1 in each constraint. We get the following permissible values:

$$\begin{aligned} w_1 = 4 - 2x_1 \quad \text{and} \quad w_1 \geq 0 \quad \text{implies} \quad x_1 \leq 2 \\ w_2 = 3 - 2x_1 \quad \text{and} \quad w_2 \geq 0 \quad \text{implies} \quad x_1 \leq 3/2 \end{aligned}$$

$$w_3 = 5 - 4x_1 \quad \text{and} \quad w_3 \geq 0 \quad \text{implies} \quad x_1 \leq 5/4$$
$$w_4 = 1 - x_1 \quad \text{and} \quad w_4 \geq 0 \quad \text{implies} \quad x_1 \leq 1$$

We pick the minimum value of x_1. This ensures that all the variables are non-negative. The basic variable corresponding to this minimum value is w_4. We get the solution as (1,0,2,1,1,0). This solution is a basic feasible solution for this iteration with x_2, w_4 being the non-basic variables while x_1, w_1, w_2, w_3 are the basic variables. Again, we have to make sure that all the basic variables are represented in terms of the non-basic variables (x_2 and w_4). To achieve this, the constraint equation $w_4 = 1 - x_1 - 5x_2$ is re-written in terms of x_2 and w_4, as $x_1 = 1 - 5x_2 - w_4$. Using this equation, we perform elementary row operations on all other equations in the previous dictionary (2.9) to eliminate x_1 and introduce w_4. For example, take the equation for Z and add two times the equation for w_4 to it, to obtain:

$$\begin{aligned} Z &= 2x_1 + x_2 \\ \underline{2w_4 &= 2 - 2x_1 - 10x_2} \\ Z &= 2 - 9x_2 - 2w_4 \end{aligned}$$

Similarly, by performing the elementary row operations on all the other equations to eliminate x_1 and introduce w_4, we get the dictionary in equation 2.10. It is noted that every variable is constrained to take a non-negative value.

$$\begin{aligned} \underline{Z &= 2 - 9x_2 - 2w_4} \\ w_1 &= 2 + 9x_2 + 2w_4 \\ w_2 &= 1 + 7x_2 + 2w_4 \\ w_3 &= 1 + 19x_2 + 4w_4 \\ x_1 &= 1 - 5x_2 - w_4 \end{aligned} \tag{2.10}$$

It can be observed that if we set all the non-basic variables to zero, we get the basic feasible solution (1,0,2,1,1,0). The value of the objective function for this solution is $Z = 2$.

We repeat the same steps. We observe the equation for Z and select a non-basic variable with a positive coefficient. Since, all the variables have negative coefficients so we cannot select any variable. Therefore the solution (1,0,2,1,1,0) is an optimal solution. The optimal value of Z is 2 which cannot be improved any further.

2.5.1 Simplex Algorithm in General form

Consider the maximization LP (equation 2.4) in the standard form after introducing the slack variables as shown in equation 2.11. This will serve as the starting dictionary. The simplex method applies to a maximization problem. If we want to solve a minimization problem using this method, we convert a minimization problem into a maximization problem. We will discuss such conversion in the next section.

$$Z = \sum_{j=1}^{n} c_j x_j \tag{2.11}$$

$$w_i = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, 2, ..., m$$

We move from one dictionary to another in search of an optimal solution by picking exactly one variable that goes from non-basic to basic and exactly one variable that performs a reverse transition. The variable that we pick to improve our objective function value is called an *entering variable*, the one which goes from non-basic to basic. The variable that is chosen to preserve the non-negativity of the current basic variables is called the *leaving variable*. This variable goes from basic to non-basic. We define B as the set of indices of the basic variables and N as the set of indices of the non-basic variables. So selecting an entering variable (x_j or w_j) implies picking j from $\{i \in N | c_i \geq 0\}$. Once the entering variable x_j is selected, the leaving variable can be picked by finding the minimum value from $\{b_i/a_{ij} \mid i \in B \text{ and } a_{ij} > 0\}$.

After picking both the entering and leaving variables, we perform elementary row operations to sort the new dictionary in terms of basic and non-basic variables. This step of

going from current dictionary to the next one is called *pivoting*. Here, we used the natural method of pivoting i.e., pick the entering variable with the maximum positive coefficient and pick the leaving variable that gives the minimum ratio. This sometimes lead to two scenarios:

1. Unboundedness: This situation occurs if all the ratios b_i/a_{ij} are negative or undefined (when any $a_{ij} = 0$) which means there is no upper bound on the value of the entering variable. This leads to an infinitely large value of the objective function. Such an LP is termed as an unbounded problem.

2. Degeneracy: This occurs when b_i is zero and we have to choose the value of entering variable to be zero. When we encounter a degenerate dictionary, there is no improvement in the value of the objective function for the subsequent dictionary. Usually, after a few degenerate dictionaries, we reach a non-degenerate dictionary that leads towards an optimal solution. There is a possibility that degeneracy leads to a dictionary encountered previously. This causes an infinite loop called cycling. To avoid this situation, we use two popular pivoting methods namely the Perturbation method and Bland's rule [36]. The first method introduces small unique perturbations in every constraint so that b_i's are non-zero. The second method picks the entering and the leaving variable with the smallest index if more than one choices are available.

Any pivoting scheme can be used to efficiently implement the simplex method. A general sketch of the steps involved in the simplex algorithm using Bland's rule for pivoting is shown in algorithm 1.

2.5.2 Choosing the initial basic feasible solution

The initial basic feasible solution can be obtained by setting all x_j's to zero if all b_i's are non-negative. If this is not the case, we use the two-phase simplex method. In two-phase simplex method, we first solve an auxiliary problem shown in 2.12 to get an initial feasible

Algorithm 1: The Simplex Algorithm

Input : A standard maximization LP
Output: An optimal solution for given LP (if one exists)

```
1  opt_solution ← false
2  ubounded ← false
3  while opt_solution = false and ubounded =false do
4      if c_j ≤ 0 for all j then
5          opt_solution ← true
6      else
7          pick that variable with smallest index j such that c_j > 0
8          if a_ij ≤ 0 for all i ∈ B then
9              unbounded ← true
10         else
11             pick the index i such that a_ij > 0 and b_i/a_ij is minimum. If there are
                 more than one such i, pick the one with smallest i.
12             perform the elementary row operations to obtain the new dictionary
13         end
14     end
15 end
```

solution. We introduce variable x_0 in every constraint. The first step is to choose the variable x_0 as the entering variable and choose the most infeasible variable, having most negative b_i value, as the leaving variable. We perform the pivoting step until we find an objective function value equal to zero [36]. The dictionary corresponding to this value will serve as our initial feasible dictionary. This part is called phase 1 of the algorithm. Phase 2 is the same as we have discussed in section 2.5.1.

$$\begin{aligned} \textit{maximize} \quad & -x_0 \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij}x_j \text{ - } x_0 \leq b_i \qquad i = 1,2,...,m \\ & x_j \geq 0 \qquad j = 0,1,...,n \end{aligned} \tag{2.12}$$

2.6 Primal Dual Method

This section will describe the second method for solving a linear program. The primal-dual method was introduced by Kuhn [27] for solving the assignment problem. This method

is also known as the Hungarian method. This method was further polished into a linear programming solver by Dantzig et al [14]. Though initially proposed as an algorithm to solve linear programs, it became popular as a method to develop combinatorial algorithms. This method was widely used to find polynomial-time approximation algorithms for NP-hard problems [18]. Wolsey [39] gave the analysis of already known algorithms for problems such as Travelling Salesman problem and the multi-dimensional Knapsack problem with the use of the primal-dual method. Even the popular Dijkstra's shortest path algorithm [11] and Ford-Fulkerson algorithm for finding the maximum flow in a network [13] can be explained as a primal-dual method. We will describe the method after going through two important properties of linear programs which follow next.

2.6.1 Duality Theory

Every linear program termed as primal comes with a dual linear program. The two problems are associated with each other via simple transformation. The primal LP can be converted into the dual LP. Conversion of that dual LP into the dual further will result in the original primal problem. Duality is an important property of linear programs which helps us analyze the behavior of many real-world problems.

Let us consider the LP 2.4 which is the primal problem. The dual of this primal can be formulated as 2.13. Here, the objective function is transformed from a maximation problem to a minimization problem. The number of dual constraints are equal to the number of primal variables (n) while the number of dual variables y_i are equal to the number of primal technological constraints (m).

$$\begin{aligned} \textit{minimize} \quad & \sum_{i=1}^{m} b_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{m} y_i a_{ij} \geq c_j \qquad j = 1,2,...,n \\ & y_i \geq 0 \qquad i = 1,2,...,m \end{aligned} \tag{2.13}$$

A feasible solution to the primal problem gives a lower bound to the primal objective function. While a feasible solution to this dual problem will provide an upper bound to the primal objective function value. The lower and upper bounds meet at a common optimal point which is termed as the optimum solution. This characteristic of primal-dual LP pair is known by the duality theorems. The *weak duality* and the *strong duality* theorems are stated below.

Theorem 2.1 (Weak Duality Theorem)**.** *If $x = (x_1, x_2, ..., x_n)$ is a feasible solution of the primal LP of (2.4) and $y = (y_1, y_2, ..., y_m)$ is a feasible solution to its dual (2.13) then*

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

Proof. This can easily be verified by using known inequalities:

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

Rearranging the summations give us:

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

Substituting the value of b_i from (2.4) will complete the proof [36].

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

□

On the other hand, the *strong duality theorem* deals with the optimality of the solution. The theorem is stated below while its proof can be found in books of Vanderbei [36] and Chvatal [7].

Theorem 2.2 (Strong Duality Theorem)**.** *If we have an optimal solution* $x = (x_1, x_2, ..., x_n)$ *to the primal LP of (2.4) and* $y = (y_1, y_2, ..., y_m)$ *is an optimal solution to its dual (2.13) then*

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

The above theorem certifies that the value of objective function of a primal problem and its dual are equal provided we have found the optimal solution of the problem. This property is very important when we are working with linear programs as it gives a certificate. These theorems also help us in determining the solvability of the linear program in consideration. We can come across two other cases. The first case is, if the primal (dual) problem is infeasible then the dual (primal) will be unbounded. While in the second case, if we have a feasible and bounded solution for the primal (dual) problem then there exist a feasible and bounded solution for the dual (primal) too [35].

2.6.2 Complementary Slackness

In this section, we will describe another property associated with the linear programs called complementary slackness. A tight constraint is the one which is satisfied at equality. If we have an optimal solution to a linear program, be it a primal or a dual problem, it is not necessary that all its constraints are tight. So, for every constraint, we have a corresponding non-negative slack variable such that if the constraint is tight, the slack variable has a zero value while for any non-tight constraint, it will have a positive value.

For the maximization problem described in (2.4), we introduce slack variables w_i as follows:

$$w_i = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, 2, ..., m$$

Similarly, for its dual in (2.13), the slack variables are:

$$z_j = \sum_{i=1}^{m} y_i a_{ij} - c_j \qquad j = 1,2,...,n$$

Now we state the *complementary slackness* theorem.

Theorem 2.3 (Complementary Slackness Theorem)**.** *If we have an optimal solution* $x = (x_1, x_2, ..., x_n)$ *to the primal LP of (2.4) and* $y = (y_1, y_2, ..., y_m)$ *is an optimal solution to its dual (2.13). Further the primal slack variables are represented by* $(w_1, w_2, ..., w_m)$ *and corresponding dual slack variables are* $(z_1, z_2, ..., z_n)$ *then*

$$x_j z_j = 0 \qquad for\ j = 1,2,...,n$$
$$w_i y_i = 0 \qquad for\ i = 1,2,...,m$$

In other words, the product of the primal variable and the corresponding dual slack variable is always zero and vice versa. The proof of this theorem can be found in Vanderbei [36]. Complementary slackness is the basis of the primal-dual schema described next.

2.6.3 Primal-Dual Schema

The primal-dual schema is a general framework for solving Linear Programs [30]. This algorithm is based on Theorem 2.3. The general idea is to use the conditions to form the *restricted primal* (RP) Problem and its dual called *dual of restricted primal* (DRP). The RP problem is formed by starting with a dual feasible solution, usually taking each dual decision variable equal to zero, and forming a set of indices of the dual constraints that are tight. Primal is assumed to be in standard form.

Theorem 2.3 states that whenever the dual constraints are not tight, the corresponding primal variables are zero in any optimal solution. Therefore, we only need to find the values of the primal decision variables that corresponds to the tight dual constraints. This is the main idea behind transforming the original problem into its restricted version (RP). This RP is further converted into its dual (DRP). We solve this DRP using a combinatorial algorithm until we get the optimal solution. The overall approach is given in figure 2.1.

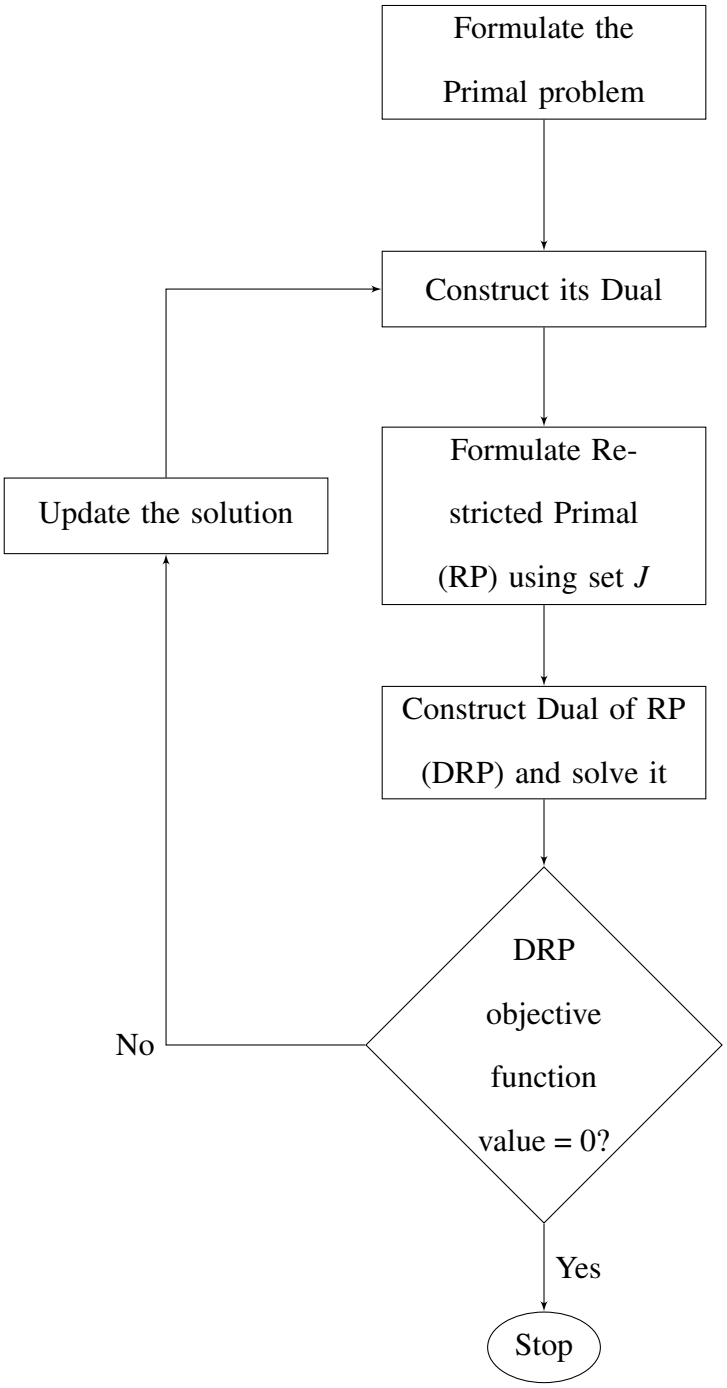

Figure 2.1: Flowchart for Primal Dual Schema

We will now describe the primal-dual method. For details, see Papadimitriou and Steiglitz [30]. We start with a minimization problem in general form as mentioned in 2.14 which is the primal problem.

$$\begin{aligned} &\textit{minimize} && cx \\ &\text{subject to} && Ax = b \\ & && x \geq 0 \end{aligned} \tag{2.14}$$

The right-hand side column vector b is assumed to be non-negative. For cases where it is less than zero, we can make it positive by multiplying -1 with the corresponding equality. The dual formulation is shown in 2.15.

$$\begin{aligned} &\textit{maximize} && \pi b \\ &\text{subject to} && \pi A \leq c \\ & && \pi \leqslant 0 \end{aligned} \tag{2.15}$$

Here it should be noted that we are using π as dual decision variable vector. The dual variables are unconstrained due to the equality constraints in the primal problem, that is, they can take both positive and negative values. If we write the complementary slackness conditions for this pair of primal-dual linear programs, we have:

$$\pi_i(A_i x - b_i) = 0 \qquad i = 1,2,...,m \tag{2.16}$$

$$(c_j - \pi A_j)x_j = 0 \qquad j = 1,2,...,n \tag{2.17}$$

Where π_i represents the i^{th} dual variable, A_i is the i^{th} row of matrix A and b_i is the right-hand side of i^{th} constraint of primal. On the other hand, x_j and c_j are the j^{th} primal variable and the coefficient respectively. A_j represents the j^{th} column of A. As we start with the standard formulation, the complementary slackness in 2.16 is always satisfied by any feasible solution. The goal is to satisfy the equation 2.17 so as to find the optimal solution (if one exists). To achieve this goal, we start with an initial feasible solution to

the dual LP and modify it iteratively until we find an optimal primal solution. The initial feasible solution π will obviously satisfy all the dual constraints. We are interested in tight constraints of which we form a set J as follows:

$$J = \{j | \pi A_j = c_j\} \tag{2.18}$$

All the constraints $j \notin J$ that are not tight will have $x_j = 0$ to satisfy optimality. For the set of constraints $j \in J$, we need to find the corresponding x_j that satisfy the condition (2.16). To find these values, we form a new LP which is called the restricted primal (RP) shown in equation 2.19. We have introduced an artificial variable w_i for every constraint i in the primal. This RP can be solved by using the ordinary simplex method defined in section 2.5.1.

$$\begin{aligned} \textit{minimize} \quad & \sum_{i=1}^{m} w_i & & \\ \text{subject to} \quad & \sum_{j \in J} a_{ij} x_j + w_i = b_i & & i = 1,2,...,m \\ & x_j \geq 0 & & j \in J \\ & x_j = 0 & & j \notin J \\ & w_i \geq 0 & & \end{aligned} \tag{2.19}$$

We construct the dual of restricted primal (DRP) as shown in 2.20.

$$\begin{aligned} \textit{maximize} \quad & \pi b & & \\ \text{subject to} \quad & \pi A_j \leq 0 & & \forall j \in J \\ & \pi_i \leq 1 & & i = 1,2,...,m \\ & \pi_i \lesseqgtr 0 & & \end{aligned} \tag{2.20}$$

The DRP is similar to the dual problem. It only includes the constraints which are in set J and upper bounds on π_i. This DRP is very important when it comes to designing

combinatorial algorithms. Many combinatorial algorithms can be explained using the DRP formulation. We can solve this DRP by using a combinatorial algorithm and denote its optimal solution as $\overline{\pi}$. If the optimal solution to the DRP is zero then we satisfy the complementary slackness conditions of Theorem 2.3 and have an optimal primal solution. If the DRP solution is of non-zero value, we improve our initial dual solution as follows:

$$\pi_{upt} = \pi + \theta\overline{\pi} \tag{2.21}$$

Where π_{upt} is the updated dual solution and θ is the improvement factor by which the current solution is to be increased. The value of θ should be selected in a way that π_{upt} will remain feasible in the dual and an improved objective function value is obtained. If we introduce b in equation 2.21, we get:

$$\pi_{upt}b = \pi b + \theta\overline{\pi}b \tag{2.22}$$

Here it can be seen clearly that to improve the new objective function value, we have to choose $\theta > 0$ as DRP objective function is already positive ($\overline{\pi} > 0$). The question now is what is the maximum value of θ we should choose. The answer to this can be found by using the following equation:

$$\pi_{upt}A_j = \pi A_j + \theta\overline{\pi}A_j \tag{2.23}$$

Our DRP solution satisfies $\overline{\pi}A_j \leq 0$ for all $j \in J$. So for all the tight constraints of dual, there is no upper limit on θ to maintain feasibility. For the remaining non-tight constraints of the dual ($j \notin J$), the LHS of equation 2.23 should obey:

$$\pi_{upt}A_j \leq c_j \tag{2.24}$$

Replacing the LHS with the equation 2.23, we get

$$\pi A_j + \theta \overline{\pi} A_j \leq c_j \tag{2.25}$$

So, for all the non-tight constraints j such that $\overline{\pi} A_j > 0$, we take the value of θ as:

$$\theta = \min_{\substack{j \notin J \\ \text{such that} \\ \overline{\pi} A_j > 0}} \left[\frac{c_j - \pi A_j}{\overline{\pi} A_j} \right] \tag{2.26}$$

We pick the minimum of all possible values of the ratio given above to ensure the feasibility of each of the dual constraint. This process will continue until the termination condition is reached i.e., the objective value of RP-DRP pair becomes zero.

If for any reason, $\overline{\pi} A_j \leq 0$ for all constraints $j \notin J$, then there is no upper bound on the θ value. In that case, the dual LP is unbounded and hence the primal is infeasible.

The interesting part typically is to derive a combinatorial algorithm for solving the DRP. For instance, in the case of shortest path problem, the DRP reduces to a path problem [30].

In the next chapter, we will define the Satisfiability problem. We will also formulate the LP models as described in this chapter for the minimum satisfiability problem.

Chapter 3

The Minimum Satisfiability problem

3.1 Introduction

One of the most studied problems in the field of computer science is the satisfiability (SAT) problem. The applications of the problem are enormous . It has a long history described in an extensive manner by Biere et al [6]. A lot of combinatorial problems in different areas of computer science are reduced to satisfiability and solved using the SAT solvers. Some notable application areas include planning in artificial intelligence [31], Electronic design automation [33], sporting leagues scheduling problems [41, 19], software verification [20], etc. A number of very effiecient SAT solvers exist now [6, 16].

In this chapter, we will give an overview of the satisfiability problem and then describe in detail, the minimum satisfiability problem. After defining the problem, we will discuss the research. Then we will formulate the problem as an ILP and described the primal-dual formulation, discussed in Section 2.6, for solving the LP relaxation. For a deeper understanding of linear programming, please refer to the excellent books [7, 36, 32].

3.2 The Satisfiability Problem

3.2.1 Definition

In mathematical logic, a *boolean* variable is a basic element that can either have a TRUE value or a FALSE value. Three basic operations that can be performed on boolean variables are $negation(\neg)$, $disjunction(\vee)$ and $conjunction(\wedge)$. The negation operation is unary while the other two operations are binary.

Literal

A *literal* is either a boolean variable x or its negation $\neg x$, also written as $\bar{x}$.

Clause

A *clause* is a disjunction of literals. Given an assignment of values TRUE/FALSE to variables, a clause is *satisfied* if it evaluates to TRUE, that is, at least one of its literals is assigned TRUE or FALSE. Otherwise, the clause is unsatisfied.

Conjunctive Normal Form (CNF)

A boolean statement which is a conjunction of clauses. A *CNF* formula is satisfied if all of its clauses are satisfied, otherwise it is unsatisfied. A *CNF* with three clauses and four variables is:

$$(x_1 \vee x_2) \wedge (\neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4)$$

3.2.2 Problem Definition

There are two versions of the satisfiability problem.

Decision Version

Given a formula in the conjunctive normal form, the goal is to find a truth assignment to boolean variables that will satisfy the formula.

Optimization Versions

Maximum Satisfiability (MAXSAT) - Given a set of clauses and a set of boolean variables, the goal is to find a truth assignment that maximizes the number of satisfied clauses.

Minimum Satisfiability (MINSAT) - Given a set of clauses and a set of the boolean variables, the objective is to find a truth assignment that minimizes the number of satisfied clauses.

3.3 The MINSAT Problem - Definition

Given a set of boolean variables X and set of clauses C as follows:

$$X = \{x_1, x_2, ..., x_n\}$$
$$C = \{c_1, c_2, ..., c_m\}$$

where each clause $c_i \in C$ is a disjunction of variables. Each x_j is either in normal form (x_j) or negated form ($\overline{x_j}$). The two literals are also referred to as positive and negative literals respectively.

The goal in the minimum satisfiability problem is to find a truth assignment of variables in X so as to satisfy the minimum number of clauses in C. A clause is satisfied when at least one of the positive literal in that clause is TRUE (boolean value 1) or at least one of the negative literal is FALSE (boolean value 0). For the rest of the thesis, we will use the binary values 1 and 0 for TRUE and FALSE respectively.

3.4 Previous Research Work

The satisfiability problem is considered to be one of the most important of all the problems in theoretical computer science with applications in a number of areas including graph theory, logic, computer science, computer engineering and operations research. A detailed history of the satisfiability problem is described by Biere et al [6]. The importance of the satisfiability problem is also due to the fact that it is the first problem shown to be NP-complete in the early 1970s by Cook [8] and Levin [28].

As we have already discussed in section 2.2.6, no polynomial time algorithms are known for solving NP-complete problems, so our goal is to find efficient approximation algorithms (section 2.2.7) with "good" performance ratio for such problems. The first of such approximation algorithm for MAXSAT was proposed by Johnson [21] which has a performance guarantee of $\frac{1}{2}$. This algorithm was based on a greedy approach for picking the literal that is used in the most number of clauses. Its value is set such that the clauses will be satisfied. We repeat until all the literals have been picked. This algorithm runs in polynomial time.

The modification to this algorithm was given by Yannakakis [40] by introducing randomization such that each literal is set to True with a probability of $\frac{1}{2}$ which also gives the same approximation ratio. Goemans and Williamson [15] improved this ratio to $\frac{3}{4}$ by using the technique of linear programming and randomized rounding. They used a relaxed version of the integer program designed for the problem and then used the rounding technique to receive the performance guarantee of $\frac{3}{4}$. There are many other contributions having somewhat same or better approximation guarantee but there is a limit on approximability which is described by Hastad [17]. His paper proved that unless $P = NP$, it is not possible to get a better approximation ratio than $\frac{7}{8}$ for MAXSAT including the MAX-3SAT, a MAXSAT instance in which each clause has at most 3 literals in it. Confirming this lower bound, Karloff and Zwick [22] gave an approximation algorithm with the guarantee of $\frac{7}{8}$ for MAX-3SAT and also prove that this ratio is tight. Another approach which seems more promising while considering the practical implications is the semi-definite programming. A survey by Anjos [2] gives a detailed review of semi-definite programming based approach for MAX-SAT.

Till now, we have discussed the previous work on the maximum satisfiability problem. Now, we change our focus to the problem in consideration in this thesis which is the minimum satisfiability problem. The minimum satisfiability problem was first introduced by Kohli et al [26]. The MINSAT problem is an NP-hard problem even if there are at most two literals in any clause. Kohli et al [26] gave a proof that MIN-2SAT, MINSAT instance when each clause contains at most 2 literals in it, and Horn-MINSAT, MINSAT instance where each clause has at most one negative literal, are NP-hard. They used an instance of 2-MAXSAT, MAXSAT instance where each clause contains exactly two literals, and transform it into a 2-MINSAT instance. The authors also proposed two approximation algorithms for solving the problem, one is a deterministic greedy algorithm while the other uses probabilistic techniques.

The greedy heuristic proposed by Kohli et al [26] is similar to the one due to Johnson

[41] for the MAXSAT problem. The proposed greedy heuristic has a worst case performance ratio of s where s is the maximum number of literals in a clause. The performance ratio for the probabilistic greedy algorithm is 2. The probabilistic greedy method uses a simple randomization technique to pick a variable and assign a boolean value to it with some probability value.

Marathe and Ravi [29] showed that MINSAT is equivalent to the vertex cover (VC) problem in the sense of approximation. They gave an approximation preserving reduction in both the directions. They also mention some of the known 2-approximation algorithms for the VC problem that can also be used to solve the MINSAT with the same performance guarantee. Dinur and Safra [12] showed that there is no approximation algorithm that can guarantee a performance ratio better than $(10\sqrt{5}-21 \backsimeq 1.3606)$ for the minimum VC problem unless $P = NP$. This implies the same result for the MINSAT problem.

A lot of work has been done on MIN-kSAT problem, MINSAT instance where each clause contain at most k literals. Notable work from Avidor and Zwick [3] gives better approximation algorithms using the semi-definite programming technique. They showed that MIN-2SAT and MIN-3SAT can be approximated within a factor of 1.1037 and 1.2136 respectively. Other notable results in their work are the inapproximability ratio for both the problems. They showed that MIN-2SAT and MIN-3SAT cannot be approximated within $\frac{15}{14}$ and $\frac{7}{6}$ respectively unless $P = NP$.

Bertsimas et al [5] used the linear programming formulation for solving the MINSAT problem. They gave an LP-based approximation algorithm for the MIN-kSAT problem with a performance guarantee of $2 - \frac{1}{2^k-1}$. They solved the LP and then rounded the fractional solution by using a randomized rounding scheme.

Our work is different from Bertsimas et al. in the sense that we have investigated the possibilities of finding approximation algorithm for MINSAT based on the primal-dual schema which is a first. Primal-dual method transforms the MINSAT LP into a DRP as discussed in Section 2.6. A combinatorial algorithm with performance ratio of 2 is known

for the vertex cover problem [4]. Such a combinatorial algorithm based on the primal-dual schema is not known for the MINSAT problem. In this thesis, We investigate the existence of such a primal-dual algorithm. In the following sections, we formulate the primal and dual LPs for the MINSAT along with the restricted primal and the dual of the restricted primal.

3.5 MINSAT as an Integer Linear Program

For transforming the MINSAT problem to a linear program, we associate a variable z_i with each clause $c_i \in C$. First, we develop the ILP then we drop the integrality constraints to obtain the LP. This ILP formulation is due to Bertsimas [5]. The objective function of this LP is to minimize the number of satisfied clauses. The values of the variables z_i and x_j are restricted to be binary, i.e. either 0 or 1.

We define two sets of indices P_i and N_i for each $c_i \in C$. The set P_i contains the indices of all the positive literals in clause c_i while N_i is the set of all the negative literal indices.

The integer linear program (ILP) for MINSAT is given as:

$$\textit{minimize} \qquad \sum_{i=1}^{m} z_i \tag{3.1}$$

$$\begin{aligned} \textit{subject to:} \quad & z_i \geq x_j && \forall j \in P_i \,,\, i = 1,2,...,m \\ & z_i \geq (1 - x_j) && \forall j \in N_i \,,\, i = 1,2,...,m \\ & z_i, x_j \in \{0,1\} \end{aligned}$$

Alternatively, we see that:

$$\begin{aligned} &\text{If } x_j = 1 \text{ then } z_i = 1 && \forall j \in P_i \,,\, i = 1,2,...,m \\ &\text{If } x_j = 0 \text{ then } z_i = 1 && \forall j \in N_i \,,\, i = 1,2,...,m \end{aligned}$$

The above ILP has a single constraint for every literal in a clause z_i which will ensure that the clause is satisfied if the positive literal has value 1 or if the negative literal has a

value 0.

3.5.1 Primal Formulation

The ILP problems take much longer to solve than the corresponding linear programs obtained by ignoring the integrality constraints [15]. It is common to solve the LP and then perform rounding to obtain the integral values. The LP is called the relaxation of the ILP. The relaxation of the above-mentioned ILP after introducing surplus variables to eliminate inequality from constraints, w_{ij} will become our primal problem as shown below:

$$\begin{aligned} \textit{minimize} \quad & \sum_{i=1}^{m} z_i \\ \textit{subject to:} \quad & z_i - x_j - w_{ij} = 0 \qquad \forall j \in P_i \,,\, i = 1,2,...,m \\ & z_i + x_j - w_{ij} = 1 \qquad \forall j \in N_i \,,\, i = 1,2,...,m \\ & z_i, x_j, w_{ij} \geq 0 \end{aligned} \tag{3.2}$$

The LP formulation presented in 3.2 is equivalent to the standard form in [30]. The surplus variable w_{ij} is defined for each clause i with literal j in it.

3.5.2 Dual Formulation

The dual of the primal problem will use the variable π_{ij} for every constraint in the primal. The subscript ij in the dual variable stands for i^{th} clause and the j^{th} literal in it. The corresponding dual problem is shown in 3.3.

$$\begin{aligned} \textit{maximize} \quad & \sum_{i} \sum_{j \in N_i} \pi_{ij} \\ \textit{subject to:} \quad & \sum_{j \in P_i \cup N_i} \pi_{ij} \leq 1 \qquad \forall i = 1,2,...m \\ & -\sum_{i | j \in P_i} \pi_{ij} + \sum_{i | j \in N_i} \pi_{ij} \leq 0 \qquad \forall j = 1,2,...n \\ & -\pi_{ij} \leq 0 \qquad \forall i, j \in P_i \cup N_i \end{aligned} \tag{3.3}$$

Any feasible solution to the primal is represented as $Z = (z_1, z_2, ..., z_m, x_1, x_2, ..., x_n)$. While a dual feasible solution is $\pi = (\pi_{i_1 j_1}, \pi_{i_2 j_2} ... \pi_{i_m j_n})$.

3.6 Primal-dual Schema for MINSAT

The primal-dual schema is one of the most used approaches in developing many combinatorial algorithms. The foremost step in executing the primal-dual algorithm is to form the restricted primal (RP) of the original problem in hand.

For constructing the RP, we start with a feasible solution π of the dual in 3.3. The initial solution is usually zero if the right-hand side for all the dual constraints is non-negative. Using this feasible π, we check which constraints are tight, left-hand side of the constraint equals right-hand side. We find set J which is the union of three sets J_1, J_2 and J_3. These sets are defined below:

$$J_1 = \left\{ \quad z_i \quad \middle| \sum_{j \in P_i \cup N_i} \pi_{ij} = 1 \right\} \tag{3.4}$$

$$J_2 = \left\{ \quad x_j \quad \middle| - \sum_{i | j \in P_i} \pi_{ij} + \sum_{i | j \in N_i} \pi_{ij} = 0 \right\} \tag{3.5}$$

$$J_3 = \left\{ \quad w_{ij} \quad \middle| - \pi_{ij} = 0 \right\} \tag{3.6}$$

The purpose of set J is to utilize Theorem 2.3 to reduce our original problem to a restricted version (RP). The complementary slackness condition states that at the point of optimality, the product of primal variables and the corresponding dual slack variables is zero and vice versa. So, if any dual constraint is not tight, then its corresponding primal variable will be zero in any optimal solution. Therefore, all those primal variables z_i, x_j and w_{ij} which are not in set J will be set to zero in any optimal solution. The restricted version of our primal problem given the set J is,

3.6.1 Restricted Primal

$$\textit{minimize} \quad \sum_{i,j} x_{ij} \tag{3.7}$$

$$\begin{aligned}
\textit{subject to:} \quad & z_i - x_j - w_{ij} + x_{ij} = 0 && \forall j \in P_i \text{ , } i = 1,2,...,m \\
& z_i + x_j - w_{ij} + x_{ij} = 1 && \forall j \in N_i \text{ , } i = 1,2,...,m \\
& z_i \geq 0 && \forall z_i \in J_1 \\
& x_j \geq 0 && \forall x_j \in J_2 \\
& w_{ij} \geq 0 && \forall w_{ij} \in J_3 \\
& z_i = 0 && \forall z_i \notin J_1 \\
& x_j = 0 && \forall x_j \notin J_2 \\
& w_{ij} = 0 && \forall w_{ij} \notin J_3 \\
& x_{ij} \geq 0 && \forall i, j \in P_i \cup N_i
\end{aligned}$$

The objective function value of the RP is denoted by Z_{RP} and x_{ij} is the artificial variable. If Z_{RP} = 0, the current dual solution is optimal and in turn the solution to the primal Z will also be optimal as the complementary slackness conditions are satisfied. If $Z_{RP} > 0$, then we have to modify the dual solution by using a solution to the dual of the restricted primal. Note that we do not compute Z_{RP} by solving the RP but by solving the DRP.

3.6.2 Dual of Restricted Primal

The dual of the restricted primal (DRP) is similar to the dual problem. Only the tight constraints, corresponding to set J, are included in it. This also imposes upper and lower bounds on the dual variables. The objective function will remain same as of the dual, that is,

$$\begin{aligned} \textit{maximize} \quad & \sum_i \sum_{j \in N_i} \pi_{ij} && (3.8) \\ \textit{subject to:} \quad & \sum_{j \in P_i \cup N_i} \pi_{ij} \leq 0 \qquad \forall z_i \in J_1 \\ & -\sum_{i|j \in P_i} \pi_{ij} + \sum_{i|j \in N_i} \pi_{ij} \leq 0 \qquad \forall x_j \in J_2 \\ & -\pi_{ij} \leq 0 \qquad \forall w_{ij} \in J_3 \\ & \pi_{ij} \leq 1 \qquad \forall i, j \in P_i \cup N_i \end{aligned}$$

The optimal solution to the DRP is represented by π_{DRP}. To modify the initial solution of the dual (π), we use π_{DRP} in the following way to update the current dual solution π.

$$\pi^* = \pi + \theta \pi_{DRP} \tag{3.9}$$

where π^* is the updated dual solution and θ is the factor by which we increase the dual solution. The value of θ is chosen such that the updated π^* will remain feasible with respect to the original dual constraints. To find an appropriate value for the θ, we have to find minimum of all the values that ensure feasibility of the corresponding dual constraints, thus we have:

$$\theta_1 = \min_{\substack{z_i \notin J_1 \\ \text{such that} \\ \sum_{j \in P_i \cup N_i} \pi_{ij_{DRP}} > 0}} \left\lceil \frac{1 - \sum_{j \in P_i \cup N_i} \pi_{ij}}{\sum_{j \in P_i \cup N_i} \pi_{ij_{DRP}}} \right\rceil \tag{3.10}$$

$$\theta_2 = \min_{\substack{x_j \notin J_2 \\ \text{such that} \\ -\sum_{i|j \in P_i} \pi_{ij_{DRP}} + \sum_{i|j \in N_i} \pi_{ij_{DRP}} > 0}} \left\lceil \frac{\sum_{i|j \in P_i} \pi_{ij} - \sum_{i|j \in N_i} \pi_{ij}}{-\sum_{i|j \in P_i} \pi_{ij_{DRP}} + \sum_{i|j \in N_i} \pi_{ij_{DRP}}} \right\rceil \tag{3.11}$$

$$\theta_3 = \min_{\substack{w_{ij} \notin J_3 \\ \text{such that} \\ -\pi_{ij_{DRP}} > 0}} \left\lceil \frac{\pi_{ij}}{-\pi_{ij_{DRP}}} \right\rceil \tag{3.12}$$

$$\theta = \min\{\theta_1, \theta_2, \theta_3\} \tag{3.13}$$

Using this updated dual solution, we again construct the sets J_1, J_2 and J_3 corresponding to the dual constraints that are tight. The restricted version of the primal is constructed again. The process is repeated until Z_{RP} is zero which ultimately gives the optimal solution to our original primal.

The DRP can, of course, be solved by simplex or any other algorithm for linear programming. Our interest is in avoiding the numerical instabilities associated with LP algorithms. Moreover, we want our algorithm for solving the DRP to be strongly polynomial and fast. Such combinatorial algorithm for solving the DRP are known for various problems [30]. No such algorithm is known for the formulation that we have proposed here for the MINSAT problem. Therefore, we would like to explore the existence of a combinatorial algorithm for solving the DRP (3.8).

3.7 Our Contributions

The primal-dual formulation presented in this chapter is an original contribution in this thesis. The ILP formulation used in this thesis is similar to Bertsimas et al [5]. The construction of the primal-dual pair and their restricted versions for the MINSAT problem is a first step to investigate the existence of a combinatorial algorithm for solving the DRP which is the focal point of this thesis.

In the next chapter, we will analyze different combinatorial methods to solve this DRP.

Chapter 4

Combinatorial Algorithms for the DRP, an Exploration

4.1 Introduction

This chapter will describe the solution strategies that we have investigated for solving the DRP of the MINSAT problem. We will present a reduction for the DRP of the MINSAT problem to a problem in bipartite graphs. We will also give two algorithms based on the reduction to solve the DRP combinatorially.

4.2 Algorithm 1: Finding a feasible path in a bipartite graph

This section will reduce our DRP into an instance of finding a feasible path in a bipartite graph. Before proceeding to the reduction, we recall from the previous chapter, the basic idea of the primal-dual method is to solve the DRP at every step and update the dual solution accordingly. Therefore, our goal is to solve the DRP combinatorially. This typically is achieved by reducing the problem into a graph problem and applying techniques such as maximum matching or maximum flow.

Let us rewrite the DRP in equation 4.1 which we have formulated in the last chapter in a simpler way by using set J instead of J_1, J_2 and J_3 and by using variable c_i instead of z_i, as we will use c_i in our algorithm. It can be seen that the DRP has three types of tight constraints, first one corresponding to z_i, second corresponding to x_j and last one corresponds to w_{ij}. We will use c_i to refer to z_i in our reduction to be consistent with the given MINSAT instance. The DRP is shown below:

$$
\begin{aligned}
&\textit{maximize} && \sum_{i}\sum_{j\in N_i}\pi_{ij} && && (4.1)\\
&\textit{subject to:} && \sum_{j\in P_i\cup N_i}\pi_{ij}\leq 0 && \forall c_i\in J\\
& && -\sum_{i|j\in P_i}\pi_{ij}+\sum_{i|j\in N_i}\pi_{ij}\leq 0 && \forall x_j\in J\\
& && -\pi_{ij}\leq 0 && \forall w_{ij}\in J\\
& && \pi_{ij}\leq 1 && \forall i,j\in P_i\cup N_i
\end{aligned}
$$

The proposed reduction will transform our DRP into an instance of bipartite graph in which we will find a $feasible\ path$.

Definition 4.1. A $feasible\ path$ is any path in a given graph that is chosen such that it satisfies the given constraints on vertices and edges.

The constraints can be restrictions on not including a particular set of edges or edges incident on a set of vertices. Before we present the reduction, we define the following:

Path: Given a graph $G=(V,E)$ with V being the set of vertices and E being the set of edges. A *path* is a walk along that graph such that we start from a vertex v and travel via edges to other vertices (other than v) such that no vertex is visited twice. For example, in figure 4.1, there are several paths such as $1\rightarrow 2\rightarrow 3\rightarrow 8$, $6\rightarrow 2\rightarrow 5\rightarrow 7$, $1\rightarrow 2\rightarrow 6$, etc.

Bipartite Graph: A graph $G=(V,E)$ is called $bipartite$ if its vertex set V can be partitioned into two disjoint subsets $V=X\cup C$ such that every edge $(u,v)\in E$ has $u\in X$ and $v\in C$. An example is shown in figure 4.2 such that $G=(X\cup C,E)$ where $X=\{1,3,5,7\}$ and $C=\{2,4,6,8\}$.

The proposed reduction is divided into three steps. We will describe the steps in detail in the subsequent subsections. The outline of the reduction is as follows:

- **Step 1:** Formulate the DRP given a basic feasible solution to the dual. Form a bipartite graph.

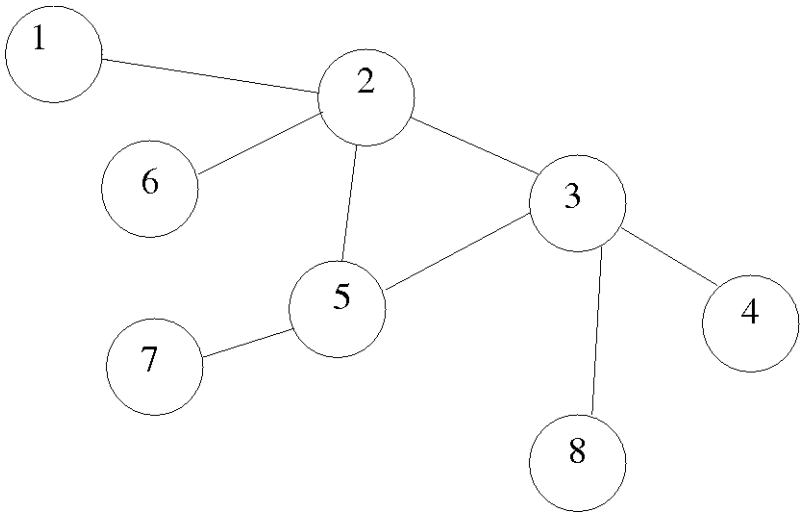

Figure 4.1: An example of a graph having 8 vertices and 8 edges

- **Step 2:** Find a forward feasible path (4.2) from the graph. Construct a feasible DRP solution from the forward path and update the dual solution. Repeat until no forward feasible path is available.

- **Step 3:** Find all the backward edges (defined in section 4.2.1) from the graph so that the objective function is improved. Output the dual solution as the final.

4.2.1 Step 1: Formulation of DRP and construction of bipartite graph

The first step in the reduction is to construct a bipartite graph from the DRP instance. This step is a one-time process such that the graph formulated will remain unchanged except for the directions of some of the edges, as described in Step 3. The step-by-step formulation is given below:

1. Start with an initial feasible dual solution π by setting all the dual variables (π_{ij}) to zero.

2. Form the set J of the primal variables whose corresponding dual constraints are tight.

3. Construct a bipartite graph $G(V,E)$ with the node set $V = X \cup C$ in the following manner:

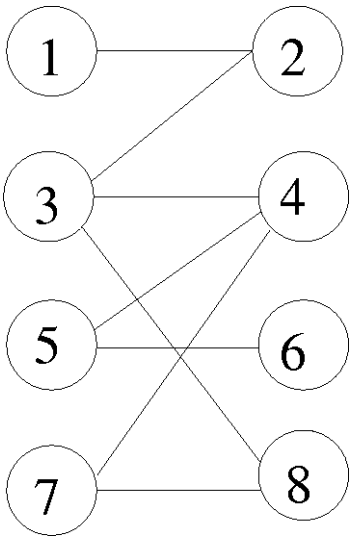

Figure 4.2: An example of a bipartite graph

- The set X contains a node for every variable $x_j \in X$ while set C contains a node for every clause $c_i \in C$.
- Now construct the edge set E by noting the type of literals in a clause c_i as follows:
  - For each positive literal x_j in clause c_i, form a forward edge (x_j, c_i)
  - For each negative literal $\bar{x}_j$ in clause c_i, form a backward edge (c_i, x_j)
  - Label each edge with its corresponding DRP variable π_{ij}. This can also be termed as the weight of the edge.

This is the end of step 1 which will give us a bipartite graph. This step can be performed in polynomial time. The running time of this step will depend upon the data structure used (see Cormen [9] for different data sructures used for representing graphs). Now we describe Step 2.

4.2.2 Step 2: Finding a forward feasible path and DRP solution

We now move to the second step for finding the forward feasible path in the bipartite graph.

Definition 4.2. In a forward feasible path, the term $forward$ is due to the fact that our path starts from a forward edge (x_j, c_i) in the graph. The term $feasible$ implies this path will give us a feasible solution that will satisfy all constraints of the DRP.

The algorithm for finding a forward feasible path is shown in Algorithm 2. In Algorithm 2, we have used *edges_aug* to represent the set of edges in the path. Initially, this set is empty. The boolean variable $path\_found$ is initialized to FALSE. It will be set to TRUE when we find such a path. Upon finding such a path, we convert it into a feasible DRP solution as below:

1. Given the set of edges in a forward feasible path, assign the value to each forward edge (x_j, c_i) in the path as $\pi_{ij} = 1$ and for each backward edge (c_i, x_j), set $\pi_{ij} = -1$. Rest of the edges not included in the path are assigned zero value. The values of π_{ij} will give us a feasible solution π_{DRP} of the current DRP.

2. Find the appropriate value of θ such that the dual problem remains feasible and update the dual solution ($\pi^* = \pi + \theta.\pi_{DRP}$).

3. By using this new dual solution π^*, construct set J again and subsequently the DRP.

4. The graph will remain the same except the directions will be reversed of all the edges for which the $\pi_i j$ =1 in current dual solution π^*

5. Repeat Algorithm 2 for finding another path until no forward path is available in the graph.

6. Go to Step 3 of the reduction.

Algorithm 2: Finding the augmenting path

Input : A bipartite graph G(X,C,E)

Output: A forward feasible path

1 $j \leftarrow 1$
2 $path\_found \leftarrow$ FALSE
3 $edges\_aug \leftarrow \emptyset$
4 **while** *(there is an unpicked edge* (x_j, c_i)*)* $\wedge$ *(*$j \leq n$*)* $\wedge$ *(*$path\_found = FALSE$*)* **do**
5 pick the edge (x_j, c_i)
6 **if** $c_i \notin J$ **then**
7 $edges\_aug \leftarrow edges\_aug \bigcup (x_j, c_i)$
8 **if** *(there is an outgoing edge* (c_i, x_k)*)* $\wedge$ *(*$w_{ik} \notin J$*)* **then**
9 $edges\_aug \leftarrow edges\_aug \bigcup (c_i, x_k)$
10 $j \leftarrow k$
11 **goto** step 4
12 **else**
13 $path\_found \leftarrow$ true
14 **end**
15 **else if** *(there is an outgoing edge* (c_i, x_k)*)* $\wedge$ *(*$w_{ik} \notin J$*)* **then**
16 $edges\_aug \leftarrow edges\_aug \bigcup \{(x_j, c_i), (c_i, x_k)\}$
17 **if** *(*x_k *has an outgoing edge* (x_k, c_l)*)* $\wedge$ *(*$c_l \notin J$*)* **then**
18 $j \leftarrow k$
19 **goto** step 4
20 **else**
21 $path\_found \leftarrow$ true
22 **else if** x_j *has another outgoing edge* (x_j, c_l) **then**
23 pick that edge
24 **goto** step 4
25 **else**
26 $j \leftarrow j + 1$
27 **end**
28 **return** $edges\_path$

In Algorithm 2, we are looking for a path starting at x_j such that it ends at any node which is not constrained. While constructing the path, the algorithm will pick a forward

edge only if the connected c_i is not constrained. A node c_i is constrained if it is present in set J. So we can only pick a forward edge it is incident to a c_i which is not present in set J. After picking the forward edge, it picks a backward edge (c_i, x_k) from c_i only if the variable w_{ik} is not in set J. If w_{ik} is in set J then π_{ik} can only take a non-negative value. For including the backward edge (c_i, x_k) in the forward feasible path, π_{ik} must take a value of -1 to satisfy the constraint on node c_i which we describe in the following paragraph.

If a node c_i is constrained, then the sum of all the incident edges, incoming (π_{ij}) and outgoing (π_{ik}), is at most zero. So, we always pick a pair of edges, one incoming from x_j and other outgoing towards any x_k such that we assign $\pi_{ij} = 1$ and $\pi_{ik} = -1$ to satisfy this constraint.

Note that the constraint on x_j is always satisfied. At a given node x_j, the constraint states that sum of incoming edges (π_{ij} due to negative literals) should be less than the sum of outgoing edges (π_{ij} due to positive literals). In any forward feasible path that we find starting from node x_j, the outgoing edge π_{ij} value of +1 will always greater than other incident edges not included in the path having value zero. Similarly, if an incoming edge incident on x_j is included in path, the π_{ij} value of -1 will always be less than zero (sum of other incident edges on x_j).

In this way, the path we obtain will always satisfy all the constraints on DRP. The algorithm is designed in a way that after each iteration, we do not need to compute θ. It will always be equal to 1 as stated in Lemma 4.3. Step 2 will terminate after a finite number of iterations which is polynomial in the number of forward edges in the graph G as stated in Lemma 4.4

Lemma 4.3. *The value of* θ *after each iteration of step 2 is equal to 1.*

Proof. Each π_{ij} can only take a value from 0,1 or -1. At each iteration of Step 2, we assign either 1 or -1 to each edge π_{ij} in the path to obtain the DRP solution. There are two cases:
Case 1: We assign 1 to π_{ij} if it is a forward edge meaning its value was zero in the previous iteration. So, the acceptable value we get for θ after using equation 3.9 is 1 which makes

the new value feasible in dual.

Case 2: We assign -1 to π_{ij} if it is a backward edge meaning its value was 1 in the previous iteration. So, the value $\theta = 1$ will actually convert it to zero so that it will not affect the feasibility of dual. □

Lemma 4.4. *Step 2 of the Algorithm 1 terminates in polynomial time.*

Proof. We know that the maximum number of forward edges in the bipartite graph G is $O(km)$ where k is the maximum number of positive literals in a clause and m is the number of clauses. At each iteration of the algorithm 2, we are finding a forward feasible path by picking an unpicked forward edge. Each forward feasible path will atleast make one clause vertex unconstrained in the next iteration. A clause vertex once constrained will not become unconstrained. In the worst case, the number of calls to algorithm 2 will be at most the number of clauses in the bipartite graph having forward edges. Therefore, the total number of iterations is $O(km)$.

□

Now, we describe the final step of the algorithm.

4.2.3 Step 3: Finding the final DRP and Dual solution

The step 2 will give us a feasible DRP solution at each iteration until we find the last forward feasible path. It is interesting to observe that the dual solution we get during each iteration is different from the previous iteration in the sense that at least one dual variable has been assigned a new value. But in terms of the dual objective function value which contain only those π_{ij}'s which are associated with backward edges, our dual solution is not improving. The reason being the assignment of -1 values to all the π_{ij}'s in the forward feasible path which are associated with the backward edges in the graph. The dual objective function only contains these π_{ij}'s, therefore, after each iteration, if any such backward edge π_{ij} is included in the feasible path, it will not increase the objective function due to its negative value. If no backward edge is included in the path, the value it gets is zero which

does not improve the value of objective function either.

This leads to the need of finding a backward feasible path that serves the purpose of improving the objective function of the dual. Once we reach the point when no forward feasible path is available, we use Algorithm 3 to find the final dual solution. This step picks all the backward edges such that the π_{ij}'s associated with them will get a value of 1. This assignment improves our objective function in each iteration until we find no further improvement. In step 6 of the Algorithm 3, we reverse the direction of the backward edge once it is picked, so that in future iterations it will not be picked again as a backward edge. This step will help us in determining the number of iterations of the algorithm in Lemma 4.5.

Algorithm 3: Backward path algorithm

Input : Dual solution $\pi = (\pi_{i_1 j_1}, \pi_{i_2 j_2} ... \pi_{i_m j_n})$ and bipartite graph $G(X, C, E)$ from Step 2

Output: Final feasible Dual solution

1 **while** *there is a backward edge* (c_i, x_j) *in G such that* $c_i, x_j \notin J$ **do**
2 pick that edge
3 set corresponding $\pi_{ij} = 1$ in π_{DRP}
4 update the dual solution π using $\theta = 1$ (Lemma 4.3)
5 find set J with the updated dual solution
6 Reverse the edge (c_i, x_j) in $G(X, C, E)$
7 **end**
8 Output the dual solution π

The running time of step 3 is polynomial in the number of backward edges in the graph G. This is stated in the following lemma,

Lemma 4.5. *Step 3 of the Algorithm 1 terminates in polynomial time.*

Proof. We know that the maximum number of backward edges in the bipartite graph G is $O(lm)$ where l is the maximum number of negative literals in a clause and m is the number of clauses. At each iteration, we pick a single backward edge. In the worst case, we pick all the backward edges so after lm number of iterations, all the backward edges are converted into forward edges which is one of the terminating conditions of the *while* loop in algorithm 3. □

Combining lemma 4.4 and 4.5, we get the following result.

Lemma 4.6. *The number of iterations in Algorithm 1 are polynomial in the number of clauses and number of variables.*

Proof. The Algorithm 1 has two major steps, finding a forward feasible path and finding a backward path. For the forward feasible path, we use the result of Lemma 4.2 that the number of iterations is bounded by number of forward edges that is $O(km)$. For the backward path, we use the result of Lemma 4.2 that the number of iterations are bounded by the number of backward edges which is $O(lm)$. Combining the two results, the number of iterations performed by the Algorithm 1 are bounded by $O(m(k+l))$. □

4.3 Algorithm 2: Edge picking in a graph

This section describes a second algorithm that we have designed for solving the DRP combinatorially. The algorithm is an extension of the previous algorithm that we have discussed. This algorithm is simpler and faster than Algorithm 1. In the previous algorithm, we look for all the feasible forward paths first and then we look for backward edges in order to improve our objective function.

The motivation for this algorithm comes from the fact that the step for finding a forward feasible path in algorithm 1 was expensive. It requires a good number of CPU processing cycles to implement due to a number of condition checks while selecting an edge to be

added in the path. In this algorithm, we have reduced many condition checks for picking an edge in the path. Our objective now is to look for a single edge instead of finding a path. Another modification we have performed in this algorithm is to combine the steps for finding a backward edge and a forward edge. The algorithm has two steps which are outlined below:

- **Step 1:** Formulate the DRP with initial feasible solution of Dual and construct an equivalent instance of a bipartite graph.
- **Step 2:** Start by finding a forward edge and construct a feasible DRP solution and update the dual. In the very next iteration, do the same with a backward edge. Repeat until no forward and backward edges are available. Output the final solution.

4.3.1 Step 1: Formulation of DRP and construction of bipartite graph

This step is the same as Step 1 of the Algorithm 1. We follow the same construction of a bipartite graph $G(X,C,E)$ from a given DRP instance. The number of nodes in the bipartite graph are $m+n$ where m and n are the number of clauses and variables respectively.

4.3.2 Step 2: Picking edges to solve DRP

This step is different from the Algorithm 1. Upon construction of the bipartite graph G, we use Algorithm 4 to find the feasible DRP solution at each iteration and finally the dual solution upon termination of the algorithm.

We start from any node x_j which has a forward edge (x_j, c_i) such that c_i is not in set J. We pick that edge and set the value 1 to its associated variable π_{ij}. This gives us a feasible DRP solution with all other π_{ij}'s being set to zero. Now, the c_i node became constrained due to $\pi ij = 1$. Therefore, we reverse the direction of this edge so that it will not be picked again as a forward edge. We then update our dual solution by taking $\theta = 1$ as per the Lemma 4.7. We reset the DRP solution to zero. We construct set J again to get the new DRP instance. The only change in graph G is to reverse the direction of the forward edge

that was picked in the last iteration. The reason being it cannot be picked again as a forward edge in future iterations, which will help us in proving Lemma 4.8.

Algorithm 4: Edge picking Algorithm

Input : Intial dual solution $\pi = (\pi_{i_1 j_1}, \pi_{i_2 j_2} ... \pi_{i_m j_n})$ and bipartite graph $G(X, C, E)$

Output: Final feasible Dual solution

```
while forward or backward edges exist do
    if forward edge (x_j, c_i) has c_i ∉ J then
        set corresponding π_ij = 1 to get feasible DRP solution π_DRP
        update the dual solution π using θ = 1 (Lemma 4.7)
        find set J with the updated dual solution
        reverse the edge (x_j, c_i) to form new G and reset π_DRP
    else if backward edge (c_i, x_j) has c_i, x_j ∉ J then
        set corresponding π_ij = 1 in π_DRP
        update the dual solution π using θ = 1
        find set J with the updated dual solution
        reverse the edge (c_i, x_j) to form new G and reset π_DRP
    else
        output the dual solution π
end
```

In the next step, we consider the updated bipartite graph G and pick a node c_i such that it has a backward edge (c_i, x_j) with both c_i and x_j not in set J. We set the associated π_{ij} to value 1 and obtain a feasible DRP solution by setting all other π_{ij}'s to zero. We repeat the steps alternatively until there are no forward and backward edges available in the graph G. This is the terminating condition of the while loop and finally we output the current dual solution as the final solution.

Lemma 4.7. *The value of* θ *after each iteration of Step 2 is equal to 1.*

Proof. At any given iteration, each π_{ij} in DRP can only take either a value of 0 or 1. If the current value of π_{ij} in DRP is zero, $\theta = 1$ will not change its updated dual value. On the other hand, if π_{ij} is picked as a forward edge, then its value in DRP is set to 1 from 0. Value of 1 for θ will update its new dual value from previous dual value of 0 to 1. For all the remaining iterations, its direction is reversed and the edge remains unpicked. This case is also true for π_{ij} if π_{ij} is picked as a backward edge.

□

Algorithm 4 also runs in polynomial time. The time complexity of the algorithm 4 is in $O(km+lm)$ as stated in Lemma 4.8. Step 1 is a one time step which constructs the initial bipartite graph.

Lemma 4.8. *The number of iterations of Algorithm 4 is polynomial in the number of clauses and has time complexity $O(km+lm)$.*

Proof. We know that the maximum number of forward edges in the bipartite graph G is $O(km)$ where k is the maximum number of positive literals in a clause and m is the number of clauses. Similarly, the maximum number of backward edges in the bipartite graph G is $O(lm)$ where l is the maximum number of negative literals. At each iteration, we first pick a forward edge and then a backward edge. So, in the worst case, the maximum number of iterations we can encounter is $O(km+lm)$. □

4.4 Our Contributions

This chapter described two of the combinatorial algorithms that we have designed to solve the DRP of the MINSAT that we constructed in Chapter 3. We have designed and analyzed around ten different strategies using the maximum flow problem, augmenting path problem, maximum matching, etc. to come up with paths in the bipartite graph. Algorithm 2 is simplification to the Algorithm 1 obtained by combining the step 2 and step 3. It is

simple in implementation and requires less processing times which will see in the next chapter.

Note that these combinatorial algorithms may not return the optimal answer to the dual. Our goal is to approximate the MINSAT problem, therefore an approximate dual solution is of interest. At this point, we have not proved any bounds on the quality of approximation for both the algorithms and we leave this an an open question for future work.

The next chapter will describe the experiments and results that we have obtained by implementing these two algorithms on given MINSAT instances.

Chapter 5

Experiments and Results

5.1 Introduction

This chapter presents the experiments that we have performed for the two algorithms on MINSAT instances from SATLIB. We start with discussing our test instances of MINSAT. We then describe the empirical results. We compare the results obtained from the two algorithms with the Simplex method.

We have used Octave 4.0.3 for conducting all the simulations presented in this chapter. The machine has an Intel Core i5 dual-core processor with a clock speed of 2.67 GHz and 6GB of RAM.

5.2 MINSAT Instances

The test instances that we have used are from SATLIB (Satisfiability Library) which is maintained by the department of Computer Science at the University of British Columbia. Each input instance file is in DIMACS cnf format. The format of the file is:

1. The file usually starts with comments. Each line of comments begins with character c.

2. After the comments end, the next line contains '$p\ cnf\ no\_of\_var\ no\_of\_clauses$' which indicates that the instance is in cnf format, no_of_var is an integer specifying the number of variables and $no\_of\_clauses$ is the exact number of clauses in it.

3. The clauses are defined next. Each clause is written in a single line as a sequence of distinct numbers separated by a space. A positive clause is represented by a positive integer which is the index of the variable. A negative clause is written as a negative number specifying the variable index. Each clause ends with an integer 0.

An example of a small input instance with 6 variables and 4 clauses in DIMACS cnf format is shown below:

```
c all comments
c will be written
c here
p cnf 6 4
1 -6 4 0
-1 -3 0
2 -4 5 6 0
-3 -4 -6 -1 -5 0
```

We have used the instances labelled "Uniform Random-3-SAT" in the library. Each instance has n variables and m clauses. The clauses are generated randomly by picking 3 literals from $2n$ possible literals (a variable and its negation).

5.3 Empirical Evaluation

We have implemented three algorithms for solving the DRP. These algorithms are the Simplex method, the Algorithm 1 and Algorithm 2. For the simplex method, we have used the GLPK (GNU Linear Programming Kit) library available for the Octave. Within this library, we use the two-phase simplex method for solving the DRP. We start with formulating the primal and dual (3.2 and 3.3) from the given instance file in DIMACS cnf format. We set all the dual variables to zero to obtain an initial feasible solution. We construct set J to form the DRP. We call the `glpk()` function to solve this DRP and obtain an optimal

solution. It is interesting to note that each `glpk()` call perform hundreds or thousands of iterations depending upon the size of the input instance. Each iteration refers to a transition from one dictionary to another, also termed as a pivoting operation. We can set the maximum limit to the number of iterations by the parameter `param.itlim`. Upon reaching the final dictionary, we get an optimal solution. This DRP solution is used to update the dual solution as in equation 3.9. We repeat the process of finding the DRP solution until the optimal value of DRP objective function becomes zero. We are assuming that the number of iterations performed by the simplex method is equal to the product of the number of `glpk()` calls and the average number of pivot operations performed in a single call.

The algorithms 1 and 2 are implemented as described in Chapter 4. We have used the following labels in Table 5.1:

instance is the name of the input instance

n is the number of variables in an input instance

m is the number of clauses in an input instance

n_c_glpk is the number of calls to `glpk()` function by the simplex method

avg_piv_glpk is the average number of pivot operations performed by a single `glpk()` call.

n_it_SIM is the total number of pivot operations perfomed by the simplex method. It is equal to the the product of the total calls to `glpk()` function (n_c_glpk) and the average number of pivot operations in a single `glpk()` call.

n_it_A1 is the total number of iterations of Algorithm 1 performed to get the final dual solution. It is the sum of iterations required in finding a forward feasible path (Step 2) and finding a backward feasible path (Step 3).

n_it_A2 is the total number of iterations of Algorithm 2 performed to get the final dual solution. In other words, it is the total number of iterations performed by Step 2 of the reduction algorithm 2.

In Table 5.1, we present the experimental results obtained by running the three algorithms on different MINSAT input instances. Not that we are assuming the worst case

performance of the simplex method in this comparison by using the average of worst and best cases. On the other hand, the number of iterations of our proposed algorithms increases somewhat linearly with the increase in the size of the input. The two plots shown in figure 5.1 and 5.2 gives a better idea of this increase in the number of iterations.

Table 5.1: Experimental results on number of iterations for the three algorithms

instance	**n**	**m**	**n_c_glpk**	**avg_piv_glpk**	**n_it_SIM**	**n_it_A1**	**n_it_A2**
Uf20-04	20	91	16	52	828	70	84
Uf20-05	20	91	13	55	714	74	86
Uf20-06	20	91	12	58	695	70	86
Uf50-01	50	172	17	80	1359	137	166
Uf50-01	50	218	27	89	2415	169	196
Uf75-01	75	161	17	84	1424	130	152
Uf75-01	75	325	25	109	2727	256	296
Uf100-01	100	430	25	136	3409	328	388
Uf125-03	125	538	35	154	5392	445	472
Uf150-013	150	645	39	162	6303	507	528
Uf175-04	175	753	36	191	6874	591	636
Uf200-02	200	860	37	206	7639	676	714

We are using the following labels in Table 5.2:

t_A1 is the processing time of the Algorithm 1. This time is the total time required by the algorithm including step 2 and step 3.

t_A2 is the processing time of the Algorithm 2.

In Table 5.2, we list the experimental results obtained in terms of the processing times of the two proposed algorithms. We have mentioned during the description of Algorithm 2 that it is somewhat faster in terms of CPU processing time. On the other hand, the number of iterations taken by Algorithm 2 is large in comparison with the Algorithm 1. The comparison plot shown in figure 5.3 will support our claim.

Although we were not able to prove a bound on the quality of the dual solution that we obtain using the proposed algorithms, but we have calculated the approximation ratio for

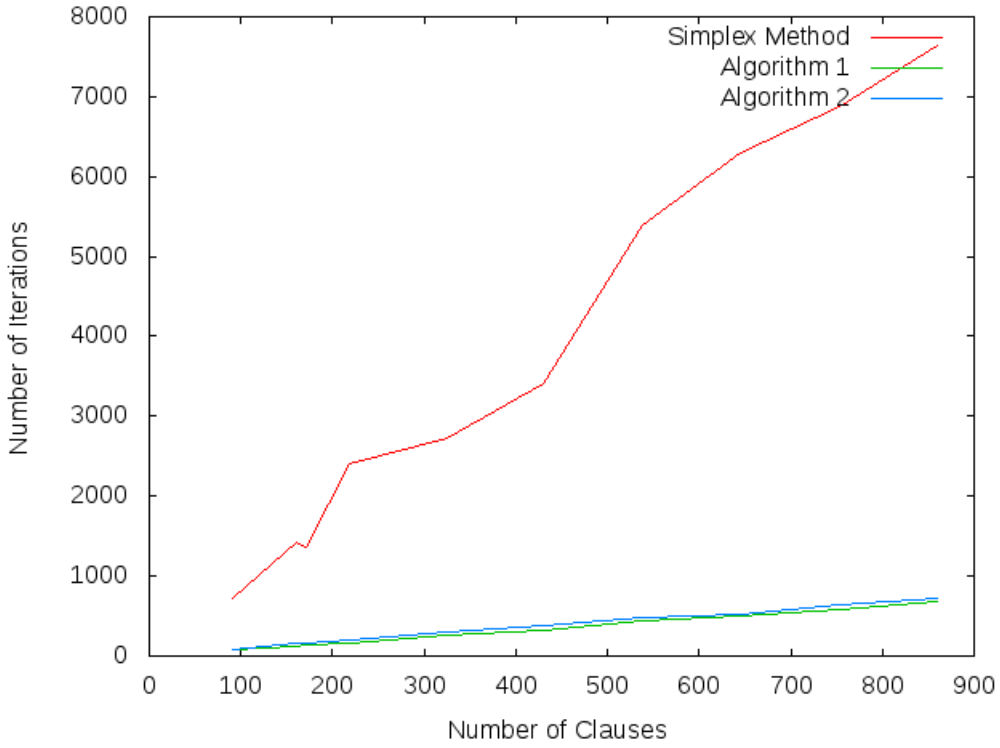

Figure 5.1: A comparison of the number of iterations of the two algorithms with the Simplex method

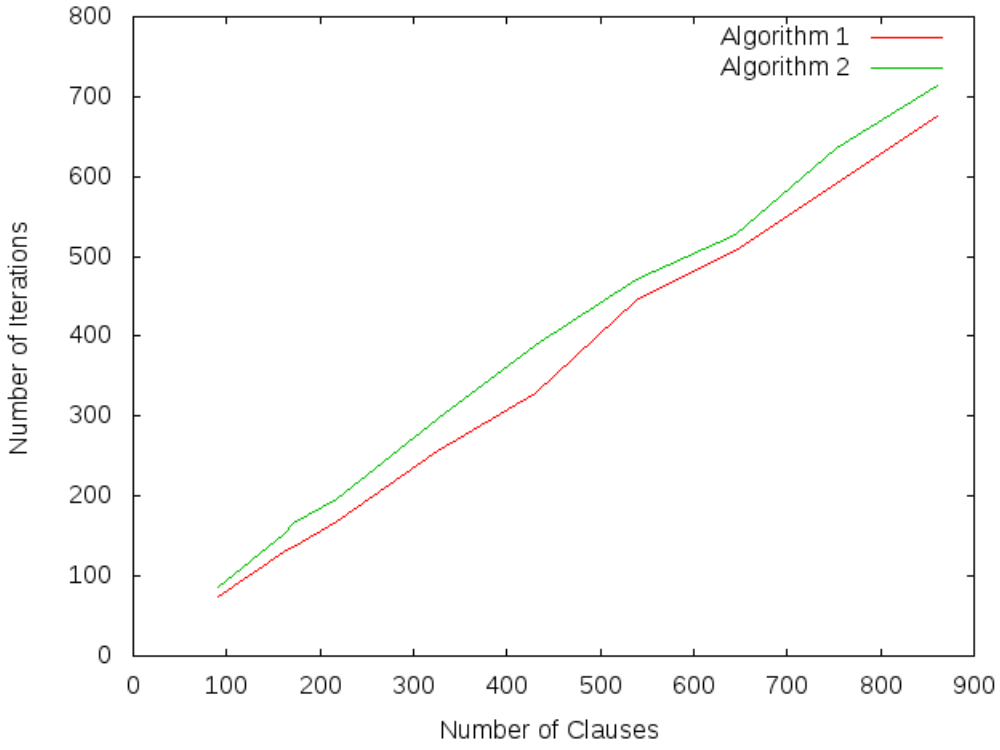

Figure 5.2: A comparison of the number of iterations between the two proposed algorithms

Table 5.2: Experimental results on the processing times of the two proposed algorithms

instance	n	m	t_A1 (sec)	t_A2 (sec)
Uf20-04	20	91	0.0671	0.0467
Uf20-05	20	91	0.0667	0.0458
Uf20-06	20	91	0.0654	0.0491
Uf50-01	50	172	0.3837	0.2145
Uf50-01	50	218	0.7292	0.6589
Uf75-01	75	161	0.3775	0.3114
Uf75-01	75	325	2.2259	1.9852
Uf100-01	100	430	8.9782	3.8951
Uf125-03	125	538	5.273	4.1748
Uf150-013	150	645	7.884	6.2653
Uf175-04	175	753	10.645	8.9562
Uf200-02	200	860	14.508	12.0123

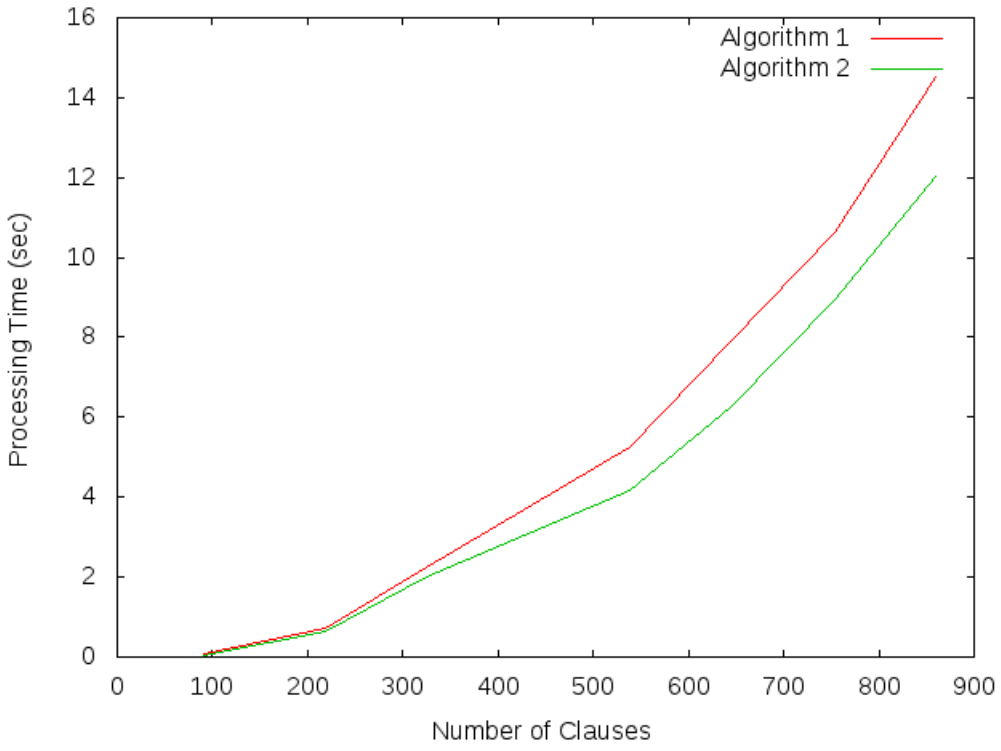

Figure 5.3: A comparison of the processing times of the two proposed algorithms

Table 5.3: Experimental results on quality of solution (Approximation Ratio) for the two algorithms

instance	n	m	obj_SIM (OPT)	obj_A1	obj_A2	AR_A1	AR_A2
Uf20-04	20	91	46.5	39	41	0.8387	0.8817
Uf20-05	20	91	49	42	42	0.8571	0.8571
Uf20-06	20	91	48	43	42	0.8958	0.875
Uf50-01	50	172	93	83	82	0.8924	0.8817
Uf50-01	50	218	112	101	97	0.9017	0.8660
Uf75-01	75	161	87	73	75	0.8390	0.8620
Uf75-01	75	325	167	145	147	0.8682	0.8802
Uf100-01	100	430	224	196	193	0.875	0.8616
Uf125-03	125	538	289	233	235	0.8062	0.8131
Uf150-013	150	645	332.5	278	263	0.8360	0.7909
Uf175-04	175	753	391.5	321	317	0.8199	0.8097
Uf200-02	200	860	453	378	356	0.8344	0.7858

the input instances by comparing it with the optimal solution obtained using the simplex method. Table 5.3 will list the dual objective function value that we obtained using the algorithm 1 and 2. We have used the simplex method directly on the dual formulation to verify the optimal value of the dual objective function. Using optimal value as a denominator and the value obtained by our algorithm as numerator, we get:

$$AR\_A1 = \frac{obj\_A1}{obj\_SIM} \tag{5.1}$$

$$AR\_A2 = \frac{obj\_A2}{obj\_SIM} \tag{5.2}$$

where,

AR_A1 is the approximation ratio for the Algorithm 1

AR_A2 is the approximation ratio for the Algorithm 2

obj_A1 is the dual objective function value obtained using Algorithm 1

obj_A2 is the dual objective function value obtained using Algorithm 2

obj_SIM= optimal value of the dual objective function obtained using the simplex method

Table 5.3 lists the experimental results obtained with respect to the approximation ratio of the two algorithms.

In the last chapter, we conclude the thesis with the summary and future research prospects.

Chapter 6

Conclusion

6.1 Summary

We have considered the minimization version of the satisfiability problem in this thesis. We have investigated the possibilities of solving the minimum satisfiability problem (MINSAT) using the primal-dual method. We have described an integer linear program of the MINSAT. We also constructed the primal, dual, restricted primal and its dual for the MINSAT problem which is the original constribution of this thesis.

We also proposed two algorithms to solve the dual of the restricted primal combinatorially. Although the dual solution may not be optimal but it gives us a way forward for the future work in this area. The algorithms we proposed are faster and requires less number of iterations to execute in comparision with the simplex method as shown in Chapter 5. The first algorithm may require less number of iterations in execution but it is somewhat slower in processing time when compared with the second algorithm.

6.2 Future Research

- We did not prove any approximation guarantee of the dual solution obtained using the two algorithms. Although we did show the value of the approximation ratio on the input data that we analyzed. It will be interesting to prove some bound on the quality of the dual solution.

- Once we prove a bound on the approximation ratio for the dual solution, we can prove a bound for the MINSAT problem.

Bibliography

[1] M. Akgul. Chapter 3. In *Topics in Relaxation and Ellipsoidal Methods, Research Notes in Mathematics*. Pitman Publishing Ltd., London, 1984.

[2] Miguel F Anjos. An improved semidefinite programming relaxation for the satisfiability problem. *Mathematical Programming*, 102(3):589–608, 2005.

[3] Adi Avidor and Uri Zwick. Approximating min 2-sat and min 3-sat. *Theory of Computing Systems*, 38(3):329–345, 2005.

[4] R Bar-Yehuda and S Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198 – 203, 1981.

[5] Dimitris Bertsimas, Chung-Piaw Teo, and Rakesh Vohra. On dependent randomized rounding algorithms. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 330–344. Springer, 1996.

[6] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009.

[7] Vasek Chvatal. *Linear Programming*. W. H. Freeman and Company, 1983.

[8] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[9] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

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

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

[12] Irit Dinur and Shmuel Safra. The importance of being biased. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 33–42. ACM, 2002.

[13] L. R. Ford and D. R. Fulkerson. *Maximal Flow Through a Network*, pages 243–248. Birkhäuser Boston, Boston, MA, 1987.

[14] L. R. Ford G. B. Dantzig and D. R. Fulkerson. A primal-dual algorithm for linear programs. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 171–181. Princeton University Press, Princeton, NJ, 1956.

[15] Michel X. Goemans and David P. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

[16] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (SAT) problem: A survey. In *Satisfiability Problem: Theory and Applications, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, March 11-13, 1996*, pages 19–152, 1996.

[17] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[18] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[19] Andrei Horbach, Thomas Bartsch, and Dirk Briskorn. Using a sat-solver to schedule sports leagues. *J. Scheduling*, 15(1):117–125, 2012.

[20] Daniel Jackson and Mandana Vaziri. Finding bugs with a constraint solver. *SIGSOFT Softw. Eng. Notes*, 25(5):14–25, August 2000.

[21] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256 – 278, 1974.

[22] Howard Karloff and Uri Zwick. A 7/8-approximation algorithm for max 3sat? In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 406–415. IEEE, 1997.

[23] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, December 1984.

[24] Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Dokl. 20*, pages 1093–1096, 1979.

[25] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[26] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani. The minimum satisfiability problem. *SIAM J. Discret. Math.*, 7(2):275–283, May 1994.

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

[28] L. A. Levin. Universal enumeration problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[29] M.V. Marathe and S.S. Ravi. On approximation algorithms for the minimum satisfiability problem. *Information Processing Letters*, 58(1):23 – 29, 1996.

[30] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[31] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artif. Intell.*, 193:45–86, December 2012.

[32] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley and Sons, 1998.

[33] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1167–1176, Sep 1996.

[34] James K. Strayer. *Linear Programming and Its Applications*. Springer, 1989.

[35] Luca Trevisan. *Combinatorial Optimization: Exact and Approximate Algorithms*. Stanford University, San Francisco, CA, 2011.

[36] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. 1996.

[37] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[38] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[39] Laurence A. Wolsey. *Heuristic analysis, linear programming and branch and bound*, pages 121–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 1980.

[40] Mihalis Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, pages 1–9, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[41] Hantao Zhang, Dapeng Li, and Haiou Shen. A sat based scheduler for tournament schedules. In *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT2004*, pages 10–13, 2004.