2015

Covering points with axis parallel lines

Jahan, Kawsar

Lethbridge, Alta : University of Lethbridge, Dept. of Mathematics and Computer Science

COVERING POINTS WITH AXIS PARALLEL LINES

KAWSAR JAHAN
Bachelor of Science, Bangladesh University of Professionals, 2009

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

COVERING POINTS WITH AXIS PARALLEL LINES

KAWSAR JAHAN

Date of Defence: December 9, 2015

Dr. Daya Gaur Co-supervisor	Professor	Ph.D.
Dr. Robert Benkoczi Co-supervisor	Associate Professor	Ph.D.
Dr. Shahadat Hossain Committee Member	Associate 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** and my **husband**.

Abstract

In this thesis, we study the problem of covering points with axis parallel lines. We named it as the point cover problem. Given a set of the n points in d dimensional space, the goal is to cover the points with minimum number of axis parallel lines. We use the iterative rounding approach which gives an integral solution to the problem. We also propose a combinatorial algorithm by using the branch and bound technique which gives an optimal integral solution to the point cover problem. Later we implemented another approximation algorithm based on primal-dual and iterative rounding approaches. Finally we show the comparative analysis between the two iterative rounding algorithms.

Acknowledgments

At the beginning, I would like to express my earnest gratitude to my supervisor and mentor Dr. Daya Gaur for his parent-like guidance, incredible supervision, inspiring motivation and brilliant suggestions throughout the period of my masters degree. I am indeed considering myself lucky to have him as my supervisor.

I would also like to convey my heartiest gratitude to my co-supervisor Dr. Robert Benkoczi for his constant endless support and helpfulness in every possible ways.

Another great hearted person I must thank is my M.Sc. supervisory committee member Dr. Shahadat Hossain who has always been there for me throughout my journey.I would also like to take the opportunity to thank my other committee member Dr. Saurya Das for his constructive suggestions and inspiration.

Besides, another big thank goes to the School of Graduate Studies and my supervisor for the financial assistance.

Last but not the least I would like to show my gratefulness to all the members of the optimization lab of University of Lethbridge for their endless support and encouragement whenever I needed it. I must want to mention some names who have significantly played important roles during this period. They are Anik Saha, Sangita Bhattacharjee, Moin Mahmud Tanvee, Mahmudun Nabi, Ram Dahal and Sina Golestanirad.

Finally I would like to thank the persons whose continuous supports and efforts have driven me to make it possible to complete my thesis are my parents, my husband and my brother.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The problem of covering points by axis parallel lines requires that a set of n points (p_1, p_2, ..., p_n) in d dimensional space has to be covered by the minimum number of axis parallel lines. We have named it as the point cover problem.

Covering points of the d dimensional surface by axis parallel lines is a topic of central interest for the researchers throughout the years. Several techniques can be used to hit points, line segments, objects like disks and rectangular boxes in 2 dimensional space by axis parallel lines [3]. Some methods exist to solve the problem of covering points in d dimensional space [2].

This thesis studies the point cover problem in the 3 dimensional space. When a problem is analysed in $3d$, it becomes more realistic because in real life, our space is 3 dimensional. There are many real life applications of the $3d$ point cover problem. For example, to get the best level of radiation in radiotherapy, the most challenging part of the job is to insert a minimum number of radio-active needles into the certain area of the body [3]. By realizing the enormous importance of covering $3d$ points with axis parallel lines, we have become interested to work in this topic.

Now we are going to talk about the contribution of our work in the field of Computer Science. We see that, our problem can be reduced to the set cover problem. The set cover problem has enormous significance in the fields of approximation and optimization algorithms. We have used our iterative rounding Algorithm 1 in computing the upper bound of

each node in our branch and bound algorithm. We are the first who have used branch and bound approach in solving point cover problems. For large input instances GLPK *IP* solver cannot compute optimal integral solution, but our branch and bound algorithm successfully computes optimal results for those instances. We hope our work will motivate to the future researchers to use different useful techniques in their research works and encourage them to explore different areas of combinatorial optimization problems.

The problem of covering points can be modelled by an integer programming solution (See Chapter 2). We design an algorithm for implementing the iterative rounding technique to obtain an integral solution of our problem. Next we design another algorithm by using the branch and bound technique to obtain the optimal integral solution of our integer programming problem. Lastly, we implement another iterative rounding algorithm and compare the results with the previous iterative rounding scheme. The algorithms are general and work for any dimension even though the experimental results have been reported for instances in $3d$.

Our first iterative rounding approach has shown better result compared to the second one. For some empirical input sets we have achieved interesting outputs for both of the approaches. In case of running time computation the second approach is better. So we found both of the iterative approaches as powerful and advantageous. The branch and bound technique which gives optimal output is also very useful and captivating.

1.2 Framework of the Thesis

In this Chapter we discuss the key terms and concepts needed. This chapter includes the concept of linear programming, integer programming and some methods to solve the linear programming problems. We also introduce the iterative rounding method, the branch and bound technique and some other important terms. After that we describe some of the related research that has been done in the field of covering points by axis parallel lines.

In Section 1.3 we give a brief description of linear programming. After that we discuss

the graphical interpretation and the primal dual method for solving linear programming problems. Next we present the simplex method to solve the linear programming problems. Lastly we mention some other methods to solve linear programming problems. In Section 1.4 we present the concept of integer programming problems. In Section 1.5 we talk about the set cover problem and we show how our problem relates to the set cover problem. We discuss the iterative rounding and the branch and bound techniques in Section 1.6 and 1.7 respectively. In Section 1.8 we discuss the work regarding the problem of covering points in R^d by axis parallel lines with minimum cardinality.

In Chapter 2 we present the point cover problem in the 3 dimensional space. We introduce the integer and the linear programming formulation of our problem. We present our iterative rounding algorithm and the branch and bound algorithm. We talk about the importance of our proposed approaches. Finally we analyse our proposed algorithms and present the results of different experiments.

Chapter 3 contains the implementation of another iterative rounding algorithm. We present the results of different experiments and the analysis of the approximation factor of the algorithm.

Finally in Chapter 4 we conclude the thesis by comparing the two iterative rounding schemes and discussing the significance of our branch and bound technique.

1.3 Linear Programming

A linear program (*LP*) is an optimization problem designed to maximize or minimize a given linear objective function by satisfying a given set of linear inequality or equality constraints. It was first introduced by Leonid Kantorovich in 1939 [12]. The variables of the linear program are relaxed, they are not restricted to be integers.

A maximization problem gives an optimal solution to the *LP* with the largest objective

function value. The canonical form of the maximization problem [14] :

$$\begin{aligned} &\text{maximize} && ax \\ &\text{subject to} && Cx \leq b \\ & && x \geq 0 \end{aligned} \tag{1.1}$$

A minimization problem gives an optimal solution to the *LP* with the smallest objective function value. The canonical form of the minimization problem :

$$\begin{aligned} &\text{minimize} && ax \\ &\text{subject to} && Cx \geq b \\ & && x \geq 0 \end{aligned} \tag{1.2}$$

The linear function ax that we want to optimize is the objective function. In the objective function, a represents the vector of known coefficients and x represents the vector of decision variables which are non-negative. $Cx \leq b$ or $Cx \geq b$ is known as the linear inequality constraint and $x \geq 0$ is known as the non-negativity constraint. C is the coefficient matrix with m rows and n columns where m is equal to the number of constraints (other than the non-negativity constraints) and n is the number of decision variables. C can also be presented as c_{ij} where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$. The vector b contains the right hand side value for each constraint.

We represent the linear programming problem in the standard form. For describing properties and algorithms for *LP*s, it is convenient to use the standard form. The transformation of an *LP* instance from canonical to standard form can be done by using a series of slack or surplus variables [11]. A linear program in the standard form is the maximization or minimization problem of a linear function subject to linear equalities. Here slack variable, s and surplus variable, s_u have used to transform the inequality constraints to equality.

The standard form of the maximization problem :

$$\begin{aligned} \text{maximize} \quad & ax \\ \text{subject to} \quad & Cx + s = b \\ & x \geq 0 \end{aligned} \tag{1.3}$$

The standard form of the minimization problem :

$$\begin{aligned} \text{minimize} \quad & ax \\ \text{subject to} \quad & Cx - s_u = b \\ & x \geq 0 \end{aligned} \tag{1.4}$$

An assignment of specific values for the decision variables is called a solution for the linear programming problem. A feasible solution is one that satisfies all the equality and inequality constraints. The feasible solution becomes an optimal solution if it has the maximum (for maximization problem) or the minimum (for minimization problem) objective function value. A problem is called infeasible if it has no feasible solution [17].

If a maximization linear program is feasible with a solution of an arbitrary large objective function value or a minimization linear program is feasible with a solution of an arbitrary small objective function value then it is called unbounded. For an unbounded maximization linear program, the objective function value is $+\infty$ and for an unbounded minimization linear program, the objective function value is $-\infty$. Finally we conclude that a linear program which is feasible and bounded has a finite optimum $(\text{maximum or minimum})$ [16].

For example, we consider a maximization problem :

$$\begin{aligned} \text{maximize} \quad & 3x_1 + 4x_2 \\ \text{subject to} \quad & 3x_1 - 4x_2 \leq 12 \\ & x_1 + 2x_2 \leq 4 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned} \tag{1.5}$$

The maximum objective function value is 12 and the values of decision variables are 4 and 0 respectively.

Now consider a minimization problem :

$$\begin{aligned} \text{minimize} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \geq 2 \\ & 2x_1 + x_2 \geq 3 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned} \tag{1.6}$$

The minimum objective function value is 1.66667 and the values of decision variables are 1.33333 and 0.333333 respectively.

To obtain the optimal objective function value we can solve a linear programming problem by using several methods including geometric interpretation, primal-dual method, simplex method and others. Optimality condition can be established by using the weak and strong duality theorems and the complementary slackness condition which are discussed later (See 1.3.2).

1.3.1 Geometric Interpretation

We can solve a linear programming problem graphically. Each of the variable values can be plotted on one of the coordinate axes. After plotting the linear constraints we obtain the feasible region. From that feasible region we compute the optimal objective function value and the optimal variable values. The feasible region is convex and the optimal solution is a vertex.

We consider our previous minimization example. Let us assume that the $x-axis$ represents the values of variable x_1 and the $y-axis$ represents the values of variable x_2. We plot $(0,1)$, $(2,0)$ points to represent the first constraint and $(0,3)$, $(\frac{3}{2},0)$ points for the second constraint respectively. Our feasible region shows that $(1.3333, 0.3333)$ is the minimum point. 1.3333 and 0.3333 values of x_1 and x_2 respectively give the minimum objective function value. Figure 1.1 of the geometric interpretation for the minimization problem is given below :

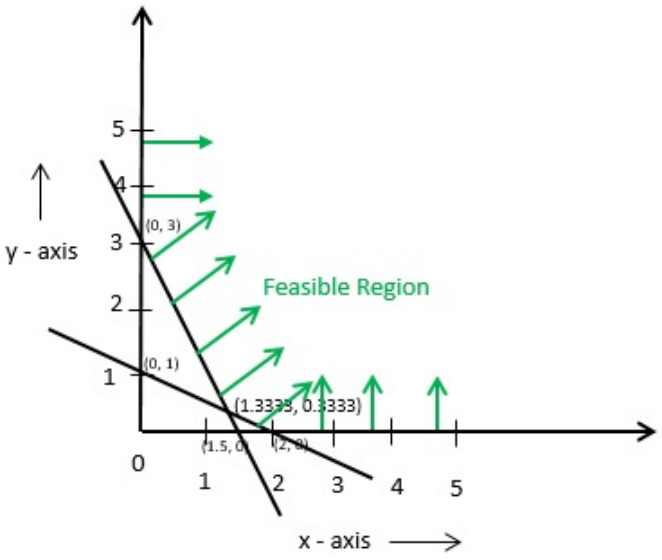

Figure 1.1: Graphical Interpretation of LP

1.3.2 Primal Dual Method

This is another method which can be used for solving linear programming problems. In 1955 Kuhn proposed his primal dual method for solving the assignment problem. We consider the linear programming problem as the primal problem. We get a dual linear program which is associated with a primal linear program [17].

Let us consider a primal linear programming problem in canonical form :

$$\begin{aligned} \text{minimize} \quad & ax \\ \text{subject to} \quad & Cx \geq b \\ & x \geq 0 \end{aligned} \tag{1.7}$$

The surplus variable, s_up for each constraint is :

$s_up = Cx - b$

the associated dual linear program is :

$$\begin{aligned} \text{maximize} \quad & by \\ \text{subject to} \quad & C^T y \leq a \\ & y \geq 0 \end{aligned} \tag{1.8}$$

The slack variable, sd for each constraint is :

$sd = a - C^T y$

The optimum of the dual provides a bound to the optimum of the primal. Here the dual (1.8) provides a lower bound to the optimum of the primal (1.7) which can be formalized as the *weak duality* theorem.

Theorem 1.1 (Weak Duality theorem)**.** *If $x = (x_1, x_2,.....,x_n)$ is a feasible and bounded solution to the primal minimization linear program* (1.7) *and $y = (y_1, y_2,.....,y_m)$ is a feasible*

and bounded solution to the dual maximization linear program (1.8) *then*

$$\sum_{j=1}^{n} a_j x_j \geq \sum_{i=1}^{m} b_i y_i \tag{1.9}$$

Where a_j and b_i are the coefficient vectors of the respective primal and dual objective functions with i = 1, 2, ..., m and j = 1, 2, ..., n.

Proof. We show from equations (1.7) and (1.8) :

$$\sum_{j=1}^{n} a_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} c_{ij} y_i \right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} c_{ij} x_j \right) y_i \geq \sum_{i=1}^{m} b_i y_i \tag{1.10}$$

which proves the theorem [17]. □

A dual solution certifies the exact value of the primal at optimality for a bounded and feasible linear program. This property is recognized as the *strong duality* theorem.

Theorem 1.2 (Strong Duality theorem)**.** *If $x = (x_1, x_2, ..., x_n)$ is an optimal solution to the primal minimization linear program* (1.7) *and $y = (y_1, y_2, ..., y_m)$ is an optimal solution to the dual maximization linear program* (1.8) *then*

$$\sum_{j=1}^{n} a_j x_j = \sum_{i=1}^{m} b_i y_i \tag{1.11}$$

For a proof of the above theorem please see Vanderbei [17]. The primal dual concept can be summarized in the following two cases [16] :

- In the case of a feasible and bounded solution of a primal problem we inevitably get a feasible and bounded solution for the dual and the other way around .
- Second case happens when the primal or the dual is unbounded, then the other is infeasible. If any of the primal or the dual is infeasible, then the other one is either infeasible or unbounded.

Now comes the *complementary slackness* theorem.

Theorem 1.3 (Complementary slackness)**.** *If $x = (x_1, x_2, ..., x_n)$ is the optimal solution to the primal* (1.7) *and $y = (y_1, y_2, ..., y_m)$ is the optimal solution to the dual* (1.8) *and let $sp = (sp_1, sp_2, ... sp_m)$ and $sd = (sd_1, sd_2, ... sd_n)$ stand for the primal slack variables and dual slack variables respectively then*

$$\begin{aligned} x_j sd_j &= 0 \quad j = 1, 2, ..., n \\ y_i sp_i &= 0 \quad i = 1, 2, ..., m \end{aligned} \tag{1.12}$$

For a proof of the theorem please see Vanderbei [17]. The complementary slackness condition is used to recover the optimal dual solution from a known optimal primal solution.

1.3.3 Simplex Method

The simplex method is one of the popular methods to solve the linear programming problem. It is an iterative process to find the optimal solution of a given linear objective function. This method was developed by George Bernard Dantzig in 1947 [1]. To apply the simplex method, a linear programming problem needs to be in standard form. The example of the maximization problem (1.5) becomes :

$$\begin{aligned} F &= 3x_1 + 4x_2 + (0)s_1 + (0)s_2 \\ s_1 &= 12 - 3x_1 + 4x_2 \\ s_2 &= 4 - x_1 - 2x_2 \end{aligned} \tag{1.13}$$

The above equation (1.13) is called dictionary. A solution of a linear programming problem with at most m non zero variables (x_1, x_2, ..., x_n, s_1, s_2, ..., s_m) of the constraint equations is a basic solution. The non-zero variables which appear on the left side of the constraint equalities are called basic or dependent variables and the right sided variables are known as non basic or independent variables. A solution becomes a basic feasible solution when all

variables are non-negative.

Now we construct the simplex tableau. This tableau consists of the augmented matrix corresponding to the constraint equations (after adding slack variables) together with the coefficients of the objective function. The initial simplex tableau of the above *LP* is :

x_1	x_2	s_1	s_2	b
3	-4	1	0	12
1	2	0	1	4
-3	-4	0	0	0

After creating the tableau we locate the most negative entry, known as the entering variable, in the bottom row. The column corresponding to the entering variable is called the entering column. In the case of ties we choose the column j where j is the smallest index.

Next we compute the ratios of the values in the column holding b-values with their corresponding positive entries in the entering column. The smallest non-negative ratio among all $\frac{b_i}{c_{ij}}$ is the leaving variable, where b_i represents the value of the i^{th} row from the $b-column$ and c_{ij} represents the value of the i^{th} row and the j^{th} column of the coefficient matrix. The row with that entry is the leaving row. In case of ties we choose the row i where i is the smallest index. We determine the pivot as the entry from the entering column and the leaving row.

In order to perform a pivot we use a series of elementary row operations (by applying Gaussian elimination method) which makes the pivot element 1 and all other entries in the entering column 0. This process ends up with a feasible solution.

As a result of the pivoting operation we move from one feasible solution to another until we arrive at the optimal solution. At that point all the entries in the bottom row of the tableau become zero or positive.

In case of a minimization problem we convert the primal problem into the dual maximization problem. In order to do this, first of all we form an augmented matrix corresponding to the inequality constraints. Then we add a last row to this augmented matrix with the

coefficients of the objective function. The augmented matrix corresponding to the example of the minimization problem (1.6) is given below :

$$\begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 3 \\ 1 & 1 & 0 \end{bmatrix}$$

In the next step we do the transpose operation in order to interchange rows and columns of the augmented matrix as follows.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 2 & 3 & 0 \end{bmatrix}$$

Here is the dual maximization problem :

$$\begin{aligned} \text{maximize} \quad & 2y_1 + 3y_2 \\ \text{subject to} \quad & y_1 + 2y_2 \leq 1 \\ & 2y_1 + y_2 \leq 1 \\ & y_1 \geq 0 \\ & y_2 \geq 0 \end{aligned} \tag{1.14}$$

We construct the initial simplex tableau for the dual (1.14) :

y_1	y_2	s_1	s_2	b
1	2	1	0	1
2	1	0	1	1
-2	-3	0	0	0

Finally we obtain the optimal solution for our minimization problem by applying the above mentioned simplex method to the dual maximization problem.

We consider a basic feasible solution for an initial dictionary if all the right-hand side values

are non-negative. If this is not the case then we introduce an auxiliary problem and solve it by using the two-phase method.

Let us consider the following auxiliary problem :

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} g_i \\ \text{subject to} \quad & \sum_{j=1}^{n} c_{ij}x_j + g_i = b_i \\ & x_j \geq 0 \qquad\qquad j = 1,2,.....,n \\ & g_i \geq 0 \qquad\qquad i = 1,2,.....,m \end{aligned} \tag{1.15}$$

Here we add or subtract a new variable for each constraint. We keep applying the steps of the simplex method to obtain a feasible dictionary for the auxiliary problem. This process is referred as phase I. After performing the phase I we follow the steps to go from a feasible solution to an optimal solution. This process is known as phase II.

At the time of performing pivot operation we may detect two special cases :

1. One case is unboundedness, where all the ratios of $\frac{b_i}{c_{ij}}$ are non-positive. In this case, the value of the entering variable can be increased indefinitely.

2. Another case is degeneracy. If the ratio of any leaving variable is $+\infty$ where the numerator is positive but the denominator vanishes then we call the pivot a degenerate pivot. Sometimes the simplex method returns to a previously generated dictionary by performing a sequence of degenerate pivot operations which causes an infinite loop. This is known as cycling. In order to avoid cycling we can use Bland's rule for pivoting. Bland's rule states that for choosing the entering and leaving variable from corresponding sets of columns and rows, respectively, we take the variable with the smallest index [17] .

Here is the algorithm for the simplex method where the linear program is in standard form and we use Bland's rule for the pivot operations.

Algorithm 1 The Simplex Algorithm

Require: A maximization LP in standard form.

Ensure: Optimal solution to the LP.

1: **while** (1) **do**
2: *set up initial simplex tableau.*
3: *determine the entering and leaving variables and the pivot.*
4: *do elementary row operations to obtain the new feasible solution and to construct the next tableau .*
5: **if** *the current feasible solution* $<$ *the previous feasible solution* **then**
6: *return the previous feasible solution.*
7: **end if**
8: **end while**
9: *return the optimal feasible solution.*

1.3.4 Some Other Methods for Solving a Linear Programming Problem

The ellipsoid method is an iterative technique which finds an optimal solution in a finite number of steps by solving a linear optimization problem with rational data. It minimizes a convex function by generating a sequence of ellipsoids whose volume uniformly decreases at every step. It does not have good practical performance. This method was first devised by Khachian in 1979 [17].

Another technique named the interior point method is also used to solve a linear convex optimization problem. Karmarkar [6] proposed this method which has better practical performance than theoretical complexity.

1.4 Integer Programming

Here we introduce the integer linear program (*ILP*).

An integer program (IP) is an optimization problem designed to maximize or minimize a

given objective function by satisfying a given set of inequality or equality constraints in which some or all of the variables are restricted to be integers.

A maximization problem gives an optimal solution to the IP with the largest integer objective function value. The canonical form of the maximization problem is :

$$\begin{aligned} \text{maximize} \quad & ax \\ \text{subject to} \quad & Cx \leq b \\ & x \in \mathbb{Z} \end{aligned} \tag{1.16}$$

A minimization problem gives an optimal solution to the IP with the smallest integer objective function value. The canonical form of the minimization problem :

$$\begin{aligned} \text{minimize} \quad & ax \\ \text{subject to} \quad & Cx \geq b \\ & x \in \mathbb{Z} \end{aligned} \tag{1.17}$$

We consider the following integer programming example for a minimization problem :

$$\begin{aligned} \text{minimize} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \geq 2 \\ & 2x_1 + x_2 \geq 3 \\ & x_1 \in \mathbb{Z} \\ & x_2 \in \mathbb{Z} \end{aligned} \tag{1.18}$$

Here the minimum objective function value is 2 and the values of decision variables are 1 and 1 respectively.

1.5 Set Cover

Given a set U of n elements and a collection of subsets $S_1, S_2 ..... S_s$ of U, the set cover problem is to find the smallest number of subsets whose union equals to U .

We consider a scenario where we want to form a team of programmers who can code C++, Ruby, Perl, Octave and Python. We have four available people including A who knows C++, Python and Perl, B who knows C++ and Ruby, C who knows Perl and Octave and D who knows C++, Octave and Ruby. We formulate the scenario as a set cover problem instance.

$U = \{C++, Ruby, Perl, Octave, Python\}$

$S_1 = \{C++, Python, Perl\}$

$S_2 = \{C++, Ruby\}$

$S_3 = \{Perl, Octave\}$

$S_4 = \{Octave, Ruby\}$

where S_1 and S_4 give the minimum set cover.

The point cover problem can be reduced to set cover where all the points are considered to form the universal set U and the points on each axis parallel line form each subset S_s. The set cover instance in d dimensions has frequency d, as d axis parallel lines go through each point.

Let us consider the following example for $d = 3$:

$U = \{P_1, P_2, P_3, P_4, P_5, P_6\}$

$S_{x1} = \{P_1, P_6\}$

$S_{x2} = \{P_2, P_4, P_5\}$

$S_{x3} = \{P_3\}$

$S_{y1} = \{P_3, P_5, P_6\}$

$S_{y2} = \{P_1, P_4\}$

$S_{y3} = \{P_2\}$

$S_{z1} = \{P_1, P_2\}$

$S_{z2} = \{P_4, P_6\}$

$S_{z3} = \{P_3, P_5\}$

Here by picking S_{x2}, S_{y1} and S_{z1} a cover is obtained.

A hypergraph, $H_G = (V, E)$ is a collection of subsets from a set, where V is a set of elements (v_1, v_2, ..., v_v) which are vertices, and E is a set of subsets (e_1, e_2 ,..., e_e) which are hyperedges. If the vertices of the hypergraph can be divided into subsets $S_1, S_2, ..., S_k$ in such a way that each edge intersects every subset at exactly one element then the hypergraph is called $k-partite$. A hypergraph with every hyperedge of size k is known as $k-uniform$. The problem of covering points with the minimum number of axis parallel lines can be reduced to the vertex covering problem in $d-partite$ hypergraph. The hypergraph H_P is constructed by considering the lines as the vertices and the lines going through each point as the edges. To demonstrate the hypergraph H_P is $d-partite$, the set of lines can be partitioned according to each of the d axes. Exactly d axis parallel lines (one line for each axis) pass across each point and each $S_i \cap e_j = 1$ where $i = 1, 2, ..., s$ and $j = 1, 2, ..., e$. The hypergraph H_P is also $d-uniform$.

1.6 Iterative Rounding

In this Section we introduce rounding. Rounding is a technique to replace a fractional value by an integer. After that we explain the concept of deterministic rounding. In deterministic rounding if a linear variable has fractional value $< \frac{1}{v}$ then the variable value is rounded to 0 and if the variable value $\geq \frac{1}{v}$ then it is rounded to 1 and here v is an arbitrary threshold variable where $v \geq 1$. Here we consider iterative rounding to be a specific version of the deterministic rounding technique.

The iterative rounding technique rounds a fractional solution of a linear programming problem and updates the current instance by reducing its size in each iteration until all the variables are rounded. Jain [5] first introduced this technique to show a 2-approximation ratio for the survivable network design problem. The iterative rounding technique works on the

extreme point solution of a linear programming problem through an iterative algorithm. It rounds a subset of variables at each iteration. Next this technique updates a current instance of the problem by replacing the number of fractional variables and passes the updated instance to the next iteration for further processing.

The iteratve rounding method follows the following steps to reach the integral solution of a linear programming problem :

1. At first the linear programming problem is formulated. One can follow the method described in Section 1.3 for formulating the linear programming problem.

2. The second step is to use the rank lemma to identify a variable with a large fractional value. Rounding the fractional value to 1 and go to the previous step.

1.7 Branch and Bound

The branch and bound method is used to find an optimal solution. This method follows a trick of throwing out large parts of the search space which does not contribute any better solution. The branch and bound technique was first proposed by Alisa Land and Alison Doig in 1960.

At the time of exploring any node of the tree this technique makes a finite decision to set one of the unbounded variables. Let us consider each variable v can have 0 or 1 value to decide the right choice. The main advantage of using the branch and bound technique is that it allows avoiding searches that do not make the solution better. For example, we consider a root node and it is going to explore. Considering $x = 0$ will lead to a solution which has upper bound ub_1 and $x = 1$ will give the solution with lower bound lb_2. We never consider the case $x = 0$ when $ub_1 < lb_2$. We easily prune all the branches where x would be set to 0. It saves both of the search space and search time. This is the reason the method to be so powerful. The following figure 1.2 shows the basic concept of branch and bound technique. In the figure, a node is explored into two subproblems by rounding $x = 0$ and $x = 1$ respectively in a binary tree. Depending on the lower and upper bounds of each

subproblem it decides which node to branch (node 3 and 4) and which node to prune (node 2 and 5) next.

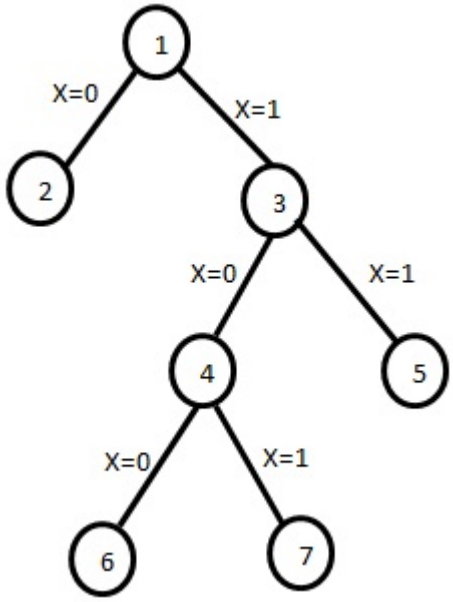

Figure 1.2: Branch and Bound Technique

Another important term is effective branching factor (EBF). Let N be the total number of nodes in the search tree and d be the depth at which the solution node was found.

$$EBF = N^{\frac{1}{d}} \tag{1.19}$$

By using the branch and bound technique, branching at only half of the levels in a binary tree is similar to search the whole tree of depth $\frac{d}{2}$. So the estimated number of nodes searched becomes as $2^{\frac{d}{2}}$. Now the EBF is :

$$EBF = (2^{\frac{d}{2}})^{\frac{1}{d}} = 2^{\frac{1}{2}} \tag{1.20}$$

For a general case of $n-ary$ tree the branching and pruning pattern gives an EBF of $\sqrt{n}$ [9] .

1.8 Some Important Terms

In this Section, we discuss the concept of the run time complexity and the approximation ratio.

1.8.1 The Time Complexity of an Algorithm

The time complexity largely depends upon the size and type of the input and the number of elementary operations performed by the algorithm. Performance of an elementary operation usually takes a fixed amount of time. We commonly express the time complexity by big O notation [15] which gives an upper bound on the running time of the algorithm. There are various classes of time complexities including constant time, linear time, polynomial time, exponential time and so on. Among all these, we discuss the polynomial time [13] and the exponential time complexities.

- An algorithm is said to be solvable in polynomial time with a running time of $O(n^k)$ which is upper bounded by a polynomial expression of the size of the input where k is a non-negative integer and n is the size of the input. Cobham and Edmonds are the inventors of the notion of polynomial time.

- An algorithm is said to be exponential time with an upper bounded running time of $O(2^{n^k})$ for a given input size where k is a non-negative integer and n is the size of the input.

In most of the cases, the simplex method has polynomial time complexity. But there are a few exceptions for which this method runs in exponential time complexity [7].
Both the ellipsoid method and the interior point method are polynomial time algorithms for solving linear programming problems.
The run time complexity of the iterative rounding algorithm depends on the solution of the linear programming problem. If the linear programming problem has a polynomial time solution then we can conclude that our iterative rounding algorithm also has a polynomial

time complexity.

Our branch and bound method can have worse time complexity if the number of nodes in the branching tree is too large.

1.8.2 Approximation Ratio

An algorithm that finds an approximate solution to a combinatorial optimization problem is known as an approximation algorithm. The approximation ratio is the ratio between the solution obtained by the approximation algorithm and the optimal solution.

$$\text{Approximation ratio} = \frac{\text{solution of the approximation algorithm}}{\text{the optimal solution}} \tag{1.21}$$

1.9 Related Research

In this Section we discuss about the previous works regarding covering points with axis parallel lines.

1.9.1 Covering Points with Axis parallel Lines

The problem definition is to cover a stated set of n points in R^d with the minimum number of axis parallel lines. The problem is formulated as an integer and a linear programming problems respectively. The formulation can be stated in the way where P_1, P_2, ..., P_n is considered as a set of n points in R^d and L is the set of axis parallel lines going through all the points. A binary variable, a_l, is considered for an integer program where a_l is 1 if the line is picked in the solution for each $l \in L$, otherwise a_l is 0.

IP :

$$\begin{aligned} &\text{minimize} && \sum_{l \in L} a_l \\ &\text{subject to} && \sum_{l : l \in L(a_i)} a_l \geq 1 \quad \forall a_i \\ & && a_l \in \{0,1\} \end{aligned} \tag{1.22}$$

By replacing the constraint $a_l \in \{0,1\}$ with $a_l \geq 0$, the integer program is relaxed to the linear program.

LP :

$$\begin{aligned} \text{minimize} \quad & \sum_{l \in L} a_l \\ \text{subject to} \quad & \sum_{l : l \in L(a_i)} a_l \geq 1 \quad \forall a_i \\ & a_l \geq 0 \end{aligned} \tag{1.23}$$

The *LP* is the primal. Now the dual is,

LP dual :

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} b_i \\ \text{subject to} \quad & \sum_{i : l \in L(a_i)} b_i \leq 1 \quad \forall l \in L \\ & b_i \geq 0 \end{aligned} \tag{1.24}$$

A vertex cover of a graph is a subset of vertices such that each edge of the graph is incident to at least one of the vertices of the subset. A matching of a graph is a subset of pairwise non-adjacent edges such that no two edges share a common vertex. In the case of the point cover problem in two dimensions, a primal feasible integer program *IP* solution is analogous to a vertex cover and its integral solution of the linear programming dual is also analogous to a matching which reminds the König-Egerváry theorem. For a proof please see [2].

Theorem 1.4 (König-Egerváry)**.** *The size of the minimum vertex cover is the same as the size of the maximum matching in a bipartite graph.*

A bipartite graph is a graph with vertices decomposed into two disjoint sets V_1 and V_2, such that every edge connects a vertex from V_1 to a vertex from V_2, and no two vertices within the same set are adjacent. A polynomial time solution can be obtained for the prob-

lem of covering points with the minimum number of axis parallel lines in two dimensions by reducing it to the vertex cover [2]. A bipartite graph can be constructed by considering each point as an edge and two axis parallel lines going through that point as the adjacent vertices of that edge. The following figure 1.3 demonstrates the construction of the bipartite graph.

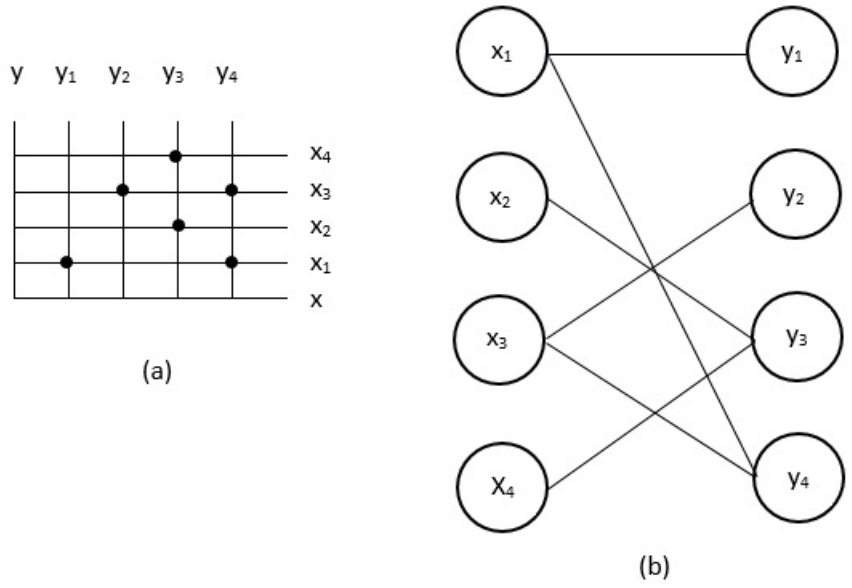

Figure 1.3: (b) A Bipartite Graph is Constructed From (a) A Point Cover Problem in 2d

According to König's theorem the optimal fractional vertex cover and the optimal integral vertex cover must have the same value for the bipartite graph which clearly demonstrates the reason the above problem is polynomially solvable.

1.9.2 Hitting Points with Straight Lines

The hitting set problem is similar to the set cover problem, where a set of objects U = *1,2,.....,o* and subsets S_j *j*=(*1,2,....,m*) are considered and the goal is to find the smallest subset $H \subseteq U$ of objects which hits every subset S_j such that $H \cap S_j \neq \emptyset$.

The greedy algorithm does not have a constant approximation ratio for the problem of covering points in the plane with the minimum number of axis parallel lines. To make the statement clear, the following example can be considered where x number of disjoint sets S_m ($m = 1, 2, ..., x$) is assumed each with $x!$ points. Each set S_m consists of $\frac{x!}{m}$ disjoint sets where each of the disjoint sets has m points. $x!$ horizontal lines cover all the points optimally, whereas the greedy algorithm may provide a solution of $x!H(x)$ vertical lines. In case of 3 disjoint sets, S_1,S_2 and S_3, each with 6 points, the optimal solution consists of 6 horizontal lines (see figure 1.4(a)) but the greedy algorithm may end up with a solution of 11 vertical lines(see figure 1.4(b)) [3].

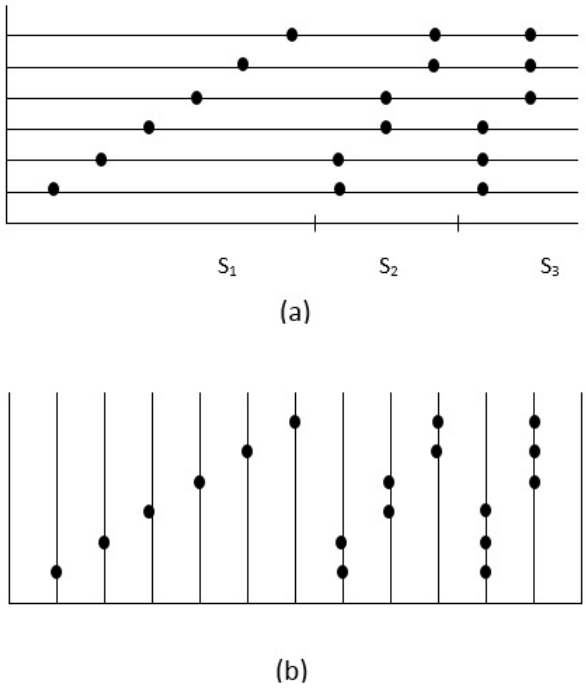

Figure 1.4: (a) The Optimal Solution Covers All the Points by 6 Lines But (b) The Greedy Solution May Choose 11 Lines

Several polynomial time approximation algorithms are observed for hitting points, vertical and horizontal line segments in two dimensional spaces.

The problem of covering points with axis parallel lines in R^3 is NP-complete if a set of k

lines can cover all the n points. For a proof please see [3].

1.9.3 Approximation Algorithms for Covering Points by Axis Parallel Lines

Several heuristic and combinatorial approaches are used to solve this problem. So some approximate and optimal results are also obtained.

A d-approximation primal dual algorithm is observed where for an uncovered point $p_i(i \in n)$ the algorithm picks all the d axis parallel lines going through that point p_i in the solution [3]. Another d-approximation deterministic rounding algorithm is also observed where the rounding process increases the value of the optimal solution at most d times [4]. The better running time of $O(nd)$ makes the primal dual based algorithm more preferable compared to the rounding algorithm.

The following theorem shows that in R^d, the point cover problem has an approximation factor of $d-1$. This theorem gives the foundation for an efficient $(d-1)$-approximation algorithm for covering points with axis parallel lines in d dimensions. For a proof of the following theorem please see [2].

Theorem 1.5. *The point cover problem in R^d can be approximated within a factor of d-1 (in $O(n^5)$ time assuming constant d)*

A dependent randomized rounding scheme is also used to obtain an integral solution of the point cover problem. At first a set of lines, S_1 is selected with decision variable values $fv_i \geq \frac{2}{d}$ and the lines with decision variable values $fv_i < \frac{2}{d}$ are placed in a set S_L. After that, the lines from S_L are partitioned according to their respective dimensions and a uniformly distributed random variable rv_i between 0 to $\frac{2}{d}$ is picked for each dimension d_i. At each dimension d_i, a set of lines with $rv_i \leq fv_i < \frac{2}{d}$ is picked to form another set S_2. The combination of the two sets S_1 and S_2 makes the cover. A $\frac{d}{2}$ approximation ratio is obtained from the above mentioned scheme [8]. Another $\frac{d}{2}$-approximation algorithm is also observed for covering points with axis parallel lines in d dimensions [10].

An integral solution of the point cover problem is also obtained which gives the foundation

for an efficient$\sqrt{d}$-approximation algorithm for covering points with axis parallel lines in d dimensions [8].

Chapter 2

Point Cover Problem

2.1 Introduction

In this Chapter we discuss the point cover problem in R^3. For a given number of points in R^3, the point cover problem determines the minimum number of axis parallel lines to cover all the points. In Section 2.2 we present the problem definition. After that we discuss the integer programming formulation for the point cover problem in Section 2.3 . In Section 2.4 we present the linear programming formulation for the point cover problem. The iterative rounding Algorithm 1 is mentioned in Section 2.5. In Section 2.6 we present a combinatorial approach which is the branch and bound algorithm to the point cover problem. We discuss an example to the problem in Section 2.7 . Finally in Section 2.8 we present the experimental results for our iterative rounding approach and branch and bound algorithm.

2.2 Problem Definition

Given a set of n points in R^d ($d \geq 3$), one has to find the minimum number of axis parallel lines to cover all the points.

We consider,

$p_1, ..., p_n$ is the set of n points in R^3.

L is the set of all the axis parallel lines going through the n points.

R^d is the dimension of space.

$L_{(p)}$ is the set of lines going through point p.

$|L| \leq nd$.

Let us consider the following three points $(n = 3)$ in R^3 :

(1,2,3), (4,3,5), (6,2,3)

Each of the above three points can be covered by any of the axis (*x-axis, y-axis or z-axis*) parallel lines respectively.

Here

$L = 9$

$L_{(1,2,3)} =$ *(*,2,3), (1,*,3), (1,2,*)*

$L_{(4,3,5)} =$ *(*,3,5), (4,*,5), (4,3,*)*

$L_{(6,2,3)} =$ *(*,2,3), (6,*,3), (6,2,*)*

Both of the points *(1,2,3)* and *(6,2,3)* can be covered by the same $x-axis$ parallel line *(*,2,3)*. We can pick any of the axis (*x-axis, y-axis or z-axis*) parallel lines to cover the point *(4,3,5)* as it does not share any common axis parallel lines with the other points. We conclude that we can cover the above three points by using two axis parallel lines. The following figure 2.1 makes the example easier to understand :

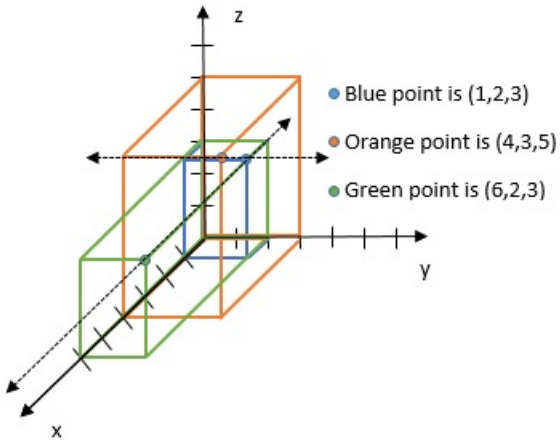

Figure 2.1: Point Cover Problem Example in 3d

2.3 Example of Reduction to Set Cover problem

In Section 1.5 we state how we can reduce the point cover problem to set cover. From the above example of Section 2.2 we show the reduction in R^3. By following the example :

$U = \{(1,2,3), (4,3,5), (6,2,3)\}$

S_{x1} = *the points on the line (*,2,3)* = $\{(1,2,3), (6,2,3)\}$

S_{y1} = *the point on the line (1,*,3)* = $\{(1,2,3)\}$

S_{z1} = *the point on the line (1,2,*)* = $\{(1,2,3)\}$

S_{x2} = *the point on the line (*,3,5)* = $\{(4,3,5)\}$

S_{y2} = *the point on the line (4,*,5)* = $\{(4,3,5)\}$

S_{z2} = *the point on the line (4,3,*)* = $\{(4,3,5)\}$

S_{y3} = *the point on the line (6,*,3)* = $\{(6,2,3)\}$

S_{z3} = *the point on the line (6,2,*)* = $\{(6,2,3)\}$

where the subsets S_{x1} and S_{y2} give the minimum set cover.

2.4 Integer Programming Formulation

In this Section we introduce the integer programming formulation for the point cover problem. A binary variable, p_l, is associated with each line $l \in L$ which receives a value 1 if the line is picked in the solution or value 0 if the line is not picked in the solution.
The *IP* to the point cover problem is :

$$\begin{aligned} \text{minimize} \quad & \sum_{l \in L} p_l \\ \text{subject to} \quad & \sum_{l : l \in L(p_i)} p_l \geq 1 \quad \forall p_i \\ & p_l \in \{0, 1\} \end{aligned} \tag{2.1}$$

The *IP* formulation restricts the decision variables to be binary. The variable p_l is associated with each line $l \in L$ and the value of p_l becomes 1 if the line is picked in the solution and 0 if the line is not picked in the solution. IP_OPT denotes the optimal integral solution for

the point cover problem.

2.5 Linear Programming Formulation

In this Section, we present the linear programming formulation.

We obtain the linear programming relaxation to the *IP* by relaxing the non-negativity constraints to $p_l \geq 0$. The *LP* formulation to the point cover problem :

$$\begin{aligned} \text{minimize} \quad & \sum_{l \in L} p_l \\ \text{subject to} \quad & \sum_{l : l \in L(p_i)} p_l \geq 1 \quad \forall p_i \\ & p_l \geq 0 \end{aligned} \tag{2.2}$$

We obtain the optimal solution to the *LP* relaxation by solving the formulated *LP*. We construct the coefficient matrix C with dimension nxL to the *LP*. Each row of C contains exactly d ones and all the other entries of L columns are exactly 0 in the row. *LP_OPT* denotes the value of the optimal solution. The *LP* solution returns the optimal values to the decision variables too.

The dual *LP* is :

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} q_i \\ \text{subject to} \quad & \sum_{i : l \in L(p_i)} q_i \leq 1 \quad \forall l \in L \\ & q_i \geq 0 \end{aligned} \tag{2.3}$$

2.6 Iterative Rounding Algorithm 1

Iterative rounding is a heuristic approach which produces an integral solution to the point cover problem. We follow the following steps to obtain an integral solution by applying the iterative rounding method :

1. At first, we solve our *LP*. The solution returns the minimum number of axis parallel lines to cover all the points and optimal values to the decision variables fractionally.

2. If our *LP* solution has no fractional variable value, then the *LP* solution is the optimal integral solution.

3. If our *LP* solution contains any fractional values of the decision variables, we round the largest fractional decision variable value to 1. In the case more than one decision variables have largest fractional value, we pick one arbitrarily.

4. After rounding the decision variable value to 1, we solve the current *LP* instance.

5. We keep following steps 1, 2, 3 and 4 until we obtain an integral solution to each of the decision variables. *IR1* denotes the value of the integral solution.

Here is our algorithm for the iterative rounding method :

Algorithm 2 Iterative Rounding Algorithm 1

Require: A minimization LP in standard form.

Ensure: Integral solution, IR1 to the LP.

while (1) **do**
  find the optimal LP solution, LP_OPT.
  if *no fractional variable,* fv_i *exists* **then**
    return LP_OPT.
  end if
  if *a fractional variable,* fv_i *exists* **then**
    find fv_i *with largest fractional value.*
    round fv_i *to 1*
  end if
end while
return *IR1 (integral solution).*

A subset of variables is rounded in each iteration of the algorithm. The current instance of the problem is updated by reducing its size based on the rounded variables and the updated instance is passed to the next iteration.

2.7 Branch and Bound Algorithm

The branch and bound technique is a combinatorial approach which produces the optimal integral solution to the point cover problem. We consider our algorithm for exploring a binary rooted tree. We follow the following steps to obtain an optimal integral solution by applying the branch and bound technique.

1. At first, we consider the value of our *LP* solution at the root node of the tree. Initially we assume an upper bound UB with the $IR1$ value and as we explore only the root node so the number of node becomes 1.

2. If the *LP* solution returns integral output for all the decision variables, we consider the LP_OPT value as the optimal integral solution.

3. On the other hand, if the *LP* solution for a node returns a fractional output of a decision variable then we explore the node to obtain two sub problems, SP_1 and SP_2. This procedure is known as branching.

    - In case of SP_1, we round the first fractional decision variable value to 0. We solve the current *LP* instance. We consider the current LP_OPT value as the lower bound, LB_1, of the current node. In order to obtain the upper bound, UB_1, of the current node we solve the current instance by using the iterative rounding algorithm. The $IR1$ value represents the upper bound. We increase the number of nodes by 1.

    - In case of SP_2, we round the first fractional decision variable value to 1. We solve the current *LP* instance. We consider the current LP_OPT value as the lower bound of the current node and name it LB_2. In order to obtain the upper bound, UB_2, of the current node, we solve the current instance by using the iterative rounding algorithm. The $IR1$ value represents the upper bound. We increase the number of nodes by 1.

The procedure of computing lower and upper bound of each node is known as bounding. After exploring the SP_1 and SP_2 we update UB with the minimum value among UB_1, UB_2 and UB.

4. In this step, we enumerate the branching and pruning methods based on the current candidate solutions in the following ways :

   - If the current value of UB is greater than the current value of LB_2, we prune the current SP_1 node and follow steps 2 and 3 to branch the SP_2 node.
   - If the current value of UB is greater than the current value of LB_1, we prune the current SP_2 node and follow steps 2 and 3 to branch the SP_1 node.
   - If the current value of UB and LB_2 is equal, we obtain the optimal integral solution LB_2, or if the current value of UB and LB_1 is equal, we obtain the optimal integral solution LB_1 at this point.
   - If none of the above condition satisfies, then the current value of UB is the optimal integral solution.

5. We keep following steps 2, 3 and 4 recursively. The recursion stops when we obtain the optimal integral solution BB_OPT and then we return the total number of nodes explored.

Here is our algorithm for the *branch and bound* technique :

Algorithm 3 Branch and Bound Algorithm

Require: A minimization LP in standard form.

Ensure: Optimal Integral solution, BB_OPT to the LP.

1: UB = inf.

2: *Numberofnodes = 1.*

3: **procedure** BBLP*($Afv \geq 1$, $fv \geq 0$, UB_cost, LB_cost)*

4: *find the optimal LP solution, LP_OPT.*

5: **if** *no fractional variable, fv_i exists* **then**

6: ***return*** *LP_OPT and total number of nodes explored.*

7: **end if**

8: **if** *a fractional variable, fv_i exists* **then**

9: *find the optimal LP solution LB_1 and integral IR solution, UB_1 by rounding fractional variable, fv_i to 0.*

10: *Numberofnodes++*

11: *find the optimal LP solution LB_2 and integral IR solution, UB_2 by rounding fractional variable, fv_i to 1.*

12: *Numberofnodes++*

13: *Update UB with the minimum value among UB_1, UB_2 and UB.*

14: **if** $UB > LB_2$ **then**

15: ***return*** BBLP*($Afv \geq 1$, $fv = 1$, UB_2, LB_2).*

16: **end if**

17: **if** $UB > LB_1$ **then**

18: ***return*** BBLP*($Afv \geq 1$, $fv = 0$, UB_1, LB_1).*

19: **end if**

20: **if** $UB = LB_2 \vee UB = LB_1$ **then**

21: **if** $UB = LB_2$ **then**

22: ***return*** *LB_2 and total number of nodes explored.*

23: **end if**

```
            if UB = LB_1 then
                return LB_1 and total number of nodes explored.
            end if
        end if
        if none of the above condition satisfies then
            return UB and total number of nodes explored.
        end if
    end if
end procedure
return BB_OPT (optimal integral solution).
```

In any real problem, the entire tree is too large. The advantage of the branch and bound technique is to avoid searching the whole tree. At each stage only the most promising node grows by estimating its bounds. Pruning is the important aspect of the branch and bound technique which cuts off and permanently discards the unprofitable branches of the tree.

2.8 An Example of the Point Cover Problem

We consider the following set of points :

(1,1,1), (1,1,2), (1,2,2), (2,2,2), (2,2,3), (2,3,3), (3,3,3), (3,3,4), (3,4,4), (1,4,4), (1,4,1)

Each of the above points can be covered with any of the axis(*x-axis, y-axis or z-axis*) parallel lines. Here,

$n = 11$

$nd = 11\text{x}3 = 33$

$L = 22$

The following figure 2.2 shows the coefficient matrix C where R and Co stands for the labels of rows and columns respectively :

Co / R	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
5	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
10	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
11	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0

Figure 2.2: Constraint Coefficient Matrix C

We obtain the *IP* solution to cover all the *n* points by axis parallel lines.

$$IP\_OPT = 6 \tag{2.4}$$

The *LP* solution to cover all the *n* points by axis parallel lines :

$$LP\_OPT = 5.5 \tag{2.5}$$

The solution produces fractional values for the decision variables. The following figure represents the decision variable values :

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	x_{22}
0.5	0	0.5	0.5	0	0.5	0	0	0	0.5	0	0.5	0	0	0.5	0.5	0.5	0	0	0.5	0	0.5

Figure 2.3: Decision Variable Values

We obtain the following integral solution by applying the iterative rounding method (from Section 2.6) :

$$IR1 = 6 \tag{2.6}$$

The approximation ratio $IR1$ becomes :

$$Approximation\ Ratio\ IR1 = \frac{IR1}{LP\_OPT} = \frac{6}{5.5} = 1.09091 \tag{2.7}$$

We obtain the following optimal integral solution by applying the branch and bound technique(from Section 2.7).

$$BB\_OPT = 6 \tag{2.8}$$

2.9 Experiments and Results

In this section, we present the empirical data obtained by running experiments on the iterative rounding Algorithm 1 and the branch and bound algorithm which computes the approximation ratio based on a 10x10x10 grid. We used a range of p values from 0.01 to 1 with a 0.01 difference. Each grid point was picked with probability p in the input instance. We repeated each experiment 25 times for each $p-value$ and took the average of 25 values. We used Octave 3.8.1 for the implementation of the algorithms. We used the GLPK solver for computing the LP value and the IP value of each input instance. All the experiments presented here were conducted on a 2.10 GHz processor with 8 GB of RAM in Ubuntu 14.04 environment.

2.9.1 Experimental Results of the Iterative Rounding Approach 1

Here we present the experimental results of the iterative rounding Algorithm 1.

- $avgIR1LP$ represents the average approximation ratio of the integral solution $IR1$ and the optimal LP solution LP_OPT.
- $avgIR1IP$ represents the average approximation ratio of the integral solution $IR1$ and the optimal IP solution IP_OPT.

- *highestIR*1*LP* represents the highest approximation ratio of the integral solution *IR*1 and the optimal *LP* solution *LP_OPT*.

- *highestIR*1*IP* represents the highest approximation ratio of the integral solution *IR*1 and the optimal *IP* solution *IP_OPT*.

Figure 2.4 summarizes the results. The first column of the table represents the $p - value$ and the second, third, fourth and fifth columns represent the average *avgIR*1*LP*, the average *avgIR*1*IP*, the *highestIR*1*LP* and the *highestIR*1*IP* data respectively.

p-value	avgIR1LP	avgIR1IP	highestIR1LP	highestIR1IP
0.01	1	1	1	1
0.02	1	1	1	1
0.03	1	1	1	1
0.04	1	1	1	1
0.05	1	1	1	1
0.06	1	1	1	1
0.07	1	1	1	1
0.08	1	1	1	1
0.09	1	1	1	1
0.1	1.00036036	1	1.00900901	1
0.11	1.00036697	1	1.00917431	1
0.12	1.00071434	1	1.00900901	1
0.13	1.00096626	1	1.01020408	1
0.14	1.00262841	1.00065574	1.01639344	1.01639344
0.15	1.0018205	1	1.01587302	1
0.16	1.00444931	1.00114496	1.0212766	1.01492537
0.17	1.00311855	1.00056338	1.02857143	1.01408451
0.18	1.00462939	1.00111894	1.01860465	1.01408451
0.19	1.00698572	1.0027568	1.02552719	1.01428571
0.2	1.00971906	1.00469315	1.04089219	1.02631579
0.21	1.01399468	1.00555617	1.04207851	1.03797468
0.22	1.01865392	1.00953585	1.04958047	1.03797468
0.23	1.01780095	1.00896775	1.05521472	1.04878049
0.24	1.01486934	1.00484996	1.0625	1.025
0.25	1.01928301	1.0076097	1.06097561	1.04819277
0.26	1.01733846	1.00712345	1.03958178	1.02439024
0.27	1.02116737	1.00887298	1.06329114	1.03703704
0.28	1.02596849	1.01217615	1.07188931	1.04597701
0.29	1.02975339	1.01390156	1.06996985	1.04444444
0.3	1.03019406	1.01558741	1.06281475	1.03448276
0.31	1.03815528	1.02045558	1.07571931	1.05494505
0.32	1.0431156	1.02393691	1.09584971	1.06521739
0.33	1.04441059	1.02609919	1.08780622	1.06521739
0.34	1.04343328	1.02501549	1.09522196	1.07446809
0.35	1.04186057	1.02436076	1.09286169	1.07291667
0.36	1.04792135	1.02701568	1.09447754	1.08163265
0.37	1.04546534	1.02729887	1.1040573	1.08163265
0.38	1.04570966	1.02788278	1.11043052	1.09090909
0.39	1.04374405	1.02843289	1.11381778	1.09
0.4	1.03442963	1.02276618	1.10859179	1.09
0.41	1.03021625	1.01923753	1.12722192	1.10416667
0.42	1.03668495	1.02509135	1.09287617	1.08
0.43	1.01929054	1.01250596	1.08986269	1.07070707
0.44	1.01709323	1.01243701	1.12346985	1.1010101
0.45	1.00838077	1.0044042	1.0421168	1.04
0.46	1.00593897	1.00436887	1.0756646	1.04901961
0.47	1.0135877	1.00915777	1.07152229	1.06060606
0.48	1.00011508	1	1.00287711	1
0.49	1.00040404	1.00040404	1.01010101	1.01010101
0.5	1.00040404	1.00040404	1.01010101	1.01010101
0.51	1	1	1	1
0.52	1	1	1	1
0.53	1.0004	1.0004	1.01	1.01
0.54	1	1	1	1
0.55	1	1	1	1
0.56	1	1	1	1
0.57	1	1	1	1
0.58	1	1	1	1
0.59	1	1	1	1
0.6	1	1	1	1
0.61	1	1	1	1
0.62	1	1	1	1
0.63	1	1	1	1
0.64	1	1	1	1
0.65	1	1	1	1
0.66	1	1	1	1
0.67	1	1	1	1
0.68	1	1	1	1
0.69	1	1	1	1
0.7	1	1	1	1

p-value	avgIR1LP	avgIR1IP	highestIR1LP	highestIR1IP
0.71	1	1	1	1
0.72	1	1	1	1
0.73	1	1	1	1
0.74	1	1	1	1
0.75	1	1	1	1
0.76	1	1	1	1
0.77	1	1	1	1
0.78	1	1	1	1
0.79	1	1	1	1
0.8	1	1	1	1
0.9	1	1	1	1
0.91	1	1	1	1
0.92	1	1	1	1
0.93	1	1	1	1
0.94	1	1	1	1
0.95	1	1	1	1
0.96	1	1	1	1
0.97	1	1	1	1
0.98	1	1	1	1
0.99	1	1	1	1
1	1	1	1	1

Figure 2.4: Computational Performance of Iterative Rounding Algorithm 1

We graphically represent the experimental results in the following figure 2.5 :

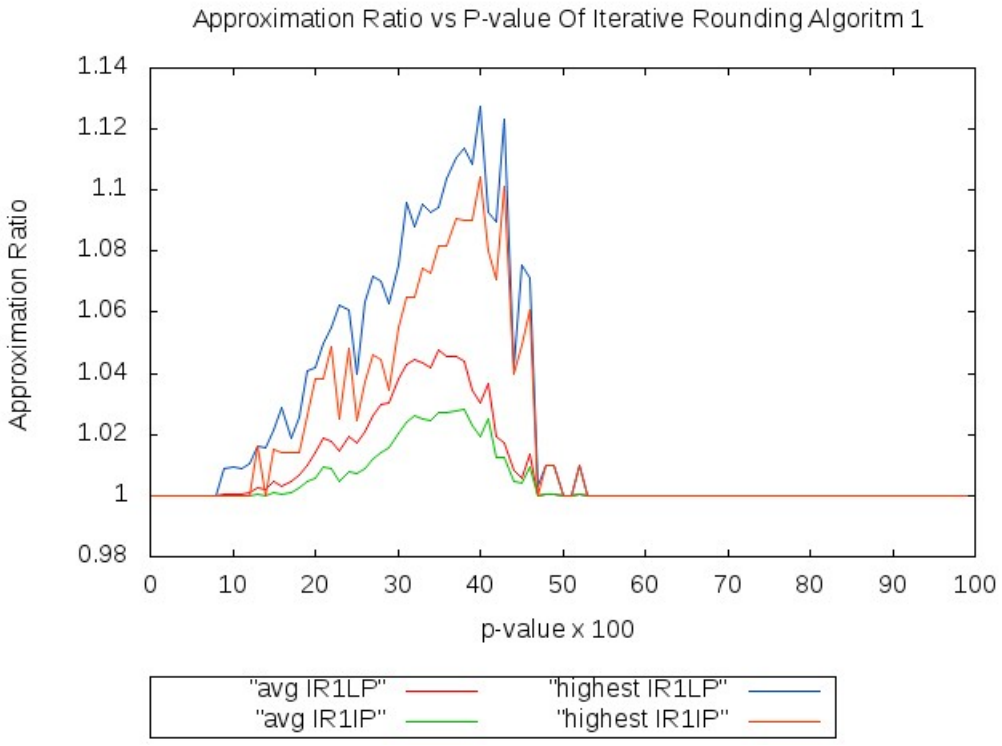

Figure 2.5: Graphical Representation of the performance of Iterative Rounding Algorithm 1

In Figure 2.5, we plot the $p-values$ along the $x-axis$ and the approximation ratio (average of 25 values and highest among 25 values) along the $y-axis$. Based on figures 2.4 and 2.5 we can say that,

- Average approximation ratio of $\frac{IR1}{LP\_OPT}$:

  In the case of the input instance of the $p-value$ from 0.01 to 0.09 we receive integral LP solutions. So the approximation ratios are 1. For the instances starting from the $p-value$ 0.1 to 0.36, the approximation ratio increases slowly and at $p = 0.41$, we obtain the highest approximation ratio which is 1.1272219. After that, from $p = 0.37$ to $p = 0.53$, the approximation ratios of the respective instances decrease slowly. From $p = 0.54$ to $p = 1$, the approximation ratios are 1.

- Average approximation ratio of $\frac{IR1}{IP\_OPT}$:

  The approximation ratio starts increasing slowly for the input instance from $p = 0.14$ to $p = 0.37$. We obtain the highest approximation ratio at $p = 0.41$ which is 1.1041667. From $p = 0.42$ the approximation ratio starts decreasing slowly and it ends up at 1 when $p = 1$.

In figure 2.6 we compare the running time of the GLPK LP solver, the GLPK IP solver and the iterative rounding function based on a 10x10x10 data grid. We plot the average time of 25 results for each $p-value$. All the times are computed in seconds.

- LP represents the average time to compute LP_OPT value.
- IP represents the average time to compute IP_OPT value.
- $IR1$ represents the average time to compute $IR1$ value.

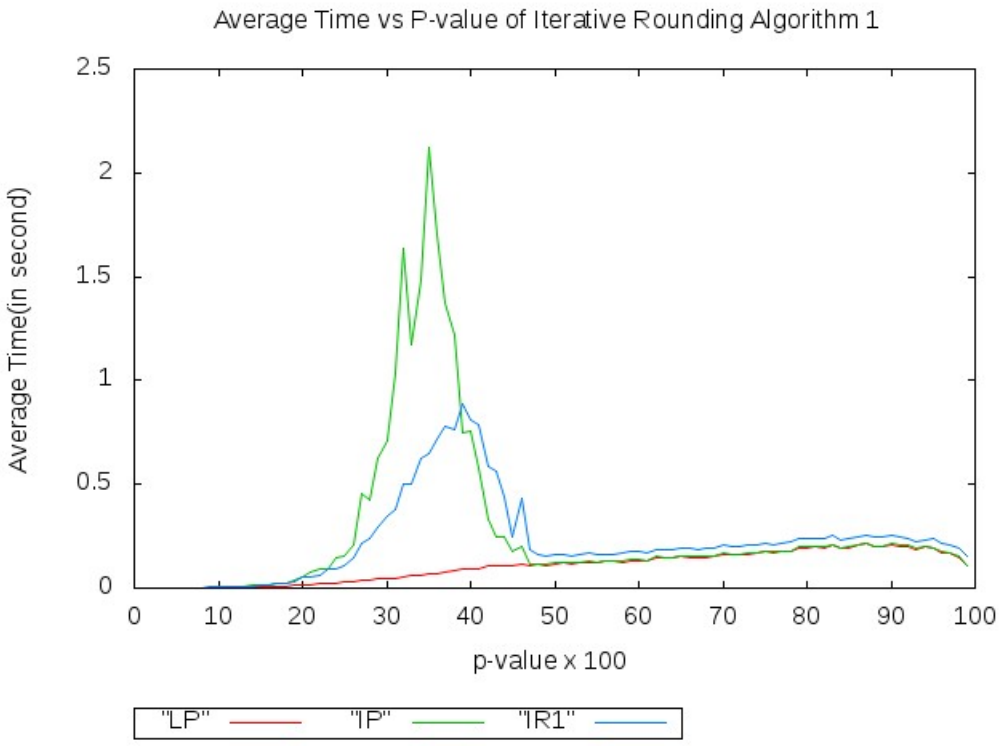

Figure 2.6: Running Time of GLPK LP Solver, GLPK IP Solver and Iterative Rounding Approach 1

Based on figure 2.6 we can say that for the input instance from $p = 0.2$ to $p = 0.4$ the iterative rounding operation shows significant improvement in running time compared to the running time of GLPK *IP* solver. So we can conclude that our iterative rounding approach performs better in computing an approximate integral solution in the case of running time.

2.9.2 Experimental Results of The Branch and Bound Approach

Here we present the experimental results of the branch and bound algorithm.

- *avgBBLP* represents the average integrality gap of the optimal integral solution *BB_OPT* and the optimal *LP* solution *LP_OPT*.
- *highestBBLP* represents the highest integrality gap of the optimal integral solution *BB_OPT* and the optimal *LP* solution *LP_OPT*.

- *nodes* represents the average total number of nodes explored for each input instance.

- *points* represents the average total number of points selected for each $p-value$.

We show the running time of the branch and bound operation based on a 10x10x10 data grid. We plot the average time of 25 results for each $p-value$. All the times are computed in second.

- *BB* represents the average time to compute BB_OPT value.

We consider three strategies for picking the fractional variable which are the first one in the ordering, the last one in the ordering and the largest one in the ordering. We get the positions of the first and the last fractional variables from the positions of the variables given by GLPK solver.

- Rounding the first fractional variable value (R_FFV) :

  Figure 2.7 summarizes the results. The first column of the table represents the $p-value$ and the second, third, fourth and fifth columns represent the average *avgBBLPF*, the *highestBBLPF*, the average $nodesF$ and the average *pointsF* data respectively.

p-value	avgBBLPF	highestBBLPF	nodesF	pointsF
0.01	1	1	1	13
0.02	1	1	1	24.72
0.03	1	1	1	34.8
0.04	1	1	1	47.08
0.05	1	1	1	57.44
0.06	1	1	1	72.28
0.07	1	1	1	77.08
0.08	1	1	1	90.12
0.09	1	1	1	101.56
0.1	1.00036036	1.00900901	1.16	115.8
0.11	1.00036697	1.00917431	1.16	122
0.12	1.00071434	1.00900901	1.56	134.12
0.13	1.00096626	1.01020408	1.4	146.04
0.14	1.00197268	1.01149425	2.28	156.48
0.15	1.0018205	1.01587302	2.6	178.2
0.16	1.00330057	1.0212766	3.16	184.08
0.17	1.00254712	1.01428571	3.72	189.04
0.18	1.00350787	1.01408451	6.76	198.2
0.19	1.00421559	1.01408451	6.28	208
0.2	1.00499686	1.01570352	7.24	226.24
0.21	1.00838674	1.02248229	30.04	237.44
0.22	1.00902051	1.02208202	30.36	241.36
0.23	1.00875546	1.02087716	51.64	252.4
0.24	1.00994557	1.0375	41.72	272
0.25	1.01157047	1.02254283	58.36	273.36
0.26	1.01012172	1.02417189	86.92	284.08
0.27	1.01218073	1.02823759	218.84	299.2
0.28	1.01358244	1.0247733	255.24	305.6
0.29	1.01560492	1.02845023	429.88	315.96
0.3	1.01435654	1.02994419	411.8	325.76
0.31	1.01727804	1.0290505	941.16	337
0.32	1.01868755	1.03384062	661.48	343.72
0.33	1.01781355	1.02817294	818.84	359.6
0.34	1.01789349	1.03825137	1507.8	366.08
0.35	1.01705099	1.02900122	595	377.68
0.36	1.02034454	1.03739514	958.68	385.6
0.37	1.01764566	1.03519757	1504.44	393.84
0.38	1.01726863	1.03299666	1031.56	409.32
0.39	1.01478371	1.03206968	2090.44	428
0.4	1.01137819	1.02642388	373	430.24
0.41	1.01063626	1.02753839	1265.8	436.44
0.42	1.01118458	1.03281939	1037.88	437
0.43	1.0066359	1.02190557	179.48	452.84
0.44	1.00449564	1.02359332	31.4	462.6
0.45	1.00395065	1.02538071	33.96	469.92
0.46	1.00151072	1.0253999	21.72	481.52
0.47	1.00430235	1.03258145	91.56	493.2
0.48	1.00011508	1.00287711	1.48	509.2
0.49	1	1	1.24	512
0.5	1	1	1.16	523.16
0.51	1	1	1.08	533.88
0.52	1	1	1	544
0.53	1	1	1.08	562.16
0.54	1	1	1.08	567.8
0.55	1	1	1	583.96
0.56	1	1	1	589
0.57	1	1	1	599.12
0.58	1	1	1	600
0.59	1	1	1	633
0.6	1	1	1	638.04
0.61	1	1	1	651.44
0.62	1	1	1	653.64
0.63	1	1	1	663.48
0.64	1	1	1	671
0.65	1	1	1	678.32
0.66	1	1	1	691.6
0.67	1	1	1	693.76
0.68	1	1	1	698.68
0.69	1	1	1	706.64
0.7	1	1	1	716.08

p-value	avgBBLPF	highestBBLPF	nodesF	pointsF
0.71	1	1	1	731.2
0.72	1	1	1	740.8
0.73	1	1	1	751.28
0.74	1	1	1	761
0.75	1	1	1	764.2
0.76	1	1	1	778
0.77	1	1	1	782.32
0.78	1	1	1	800.64
0.79	1	1	1	814.36
0.8	1	1	1	822
0.81	1	1	1	837.68
0.82	1	1	1	839
0.83	1	1	1	861.48
0.84	1	1	1	865
0.85	1	1	1	868.72
0.86	1	1	1	876.96
0.87	1	1	1	886
0.88	1	1	1	894.88
0.89	1	1	1	904.08
0.9	1	1	1	911.8
0.91	1	1	1	923.56
0.92	1	1	1	930.68
0.93	1	1	1	941.28
0.94	1	1	1	950.96
0.95	1	1	1	959.32
0.96	1	1	1	971.64
0.97	1	1	1	979.16
0.98	1	1	1	988.44
0.99	1	1	1	996
1	1	1	1	1000

Figure 2.7: Computational Performance of Branch and Bound Algorithm with R_FFV

We graphically represent the experimental summary of the branch and bound algorithm with R_FFV in the following figure 2.8 :

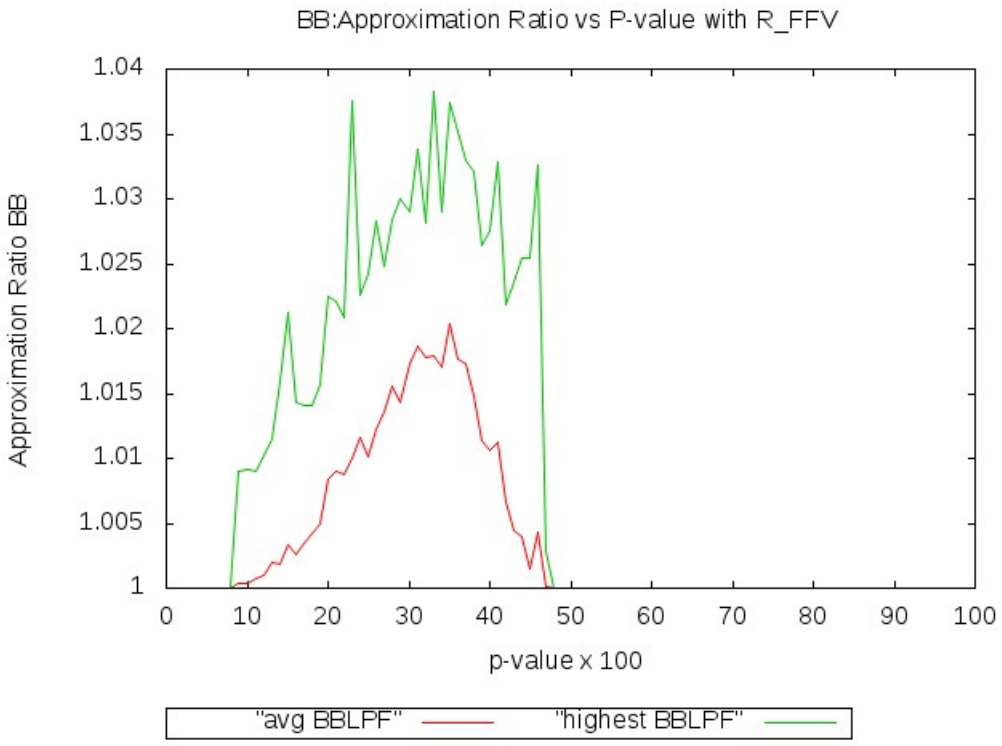

Figure 2.8: Graphical Representation of the Performance of Branch and Bound Algorithm with R_FFV

In figure 2.9 we plot the $p-values$ along the $x-axis$ and the approximation ratio(average of 25 values and highest among 25 values) along the $y-axis$. By observing both the table and the plot we can say that,

- Average approximation ratio of $\frac{BB\_OPT}{LP\_OPT}$:

  The approximation ratio starts increasing slowly for the input instance from $p = 0.1$ to $p = 0.36$. We obtain the highest approximation ratio at $p = 0.34$ which is 1.03825137. From $p = 0.37$, the approximation ratio starts decreasing slowly and it ends up at 1.

In figure 2.9 we plot the $p-values$ along the $x-axis$ and the $nodesF$ and the $pointsF$(average of 25 values) along the $y-axis$. Based on figure 2.9 we can say

that, the *nodes* grows polynomially with the input instance starting from $p = 0.1$ to $p = 0.29$. At $p = 0.39$ we obtain the highest number of nodes explored , which is 2090.44 to cover the 428 input points . From $p = 0.41$, the number of nodes starts decreasing slowly and it ends up at 1.

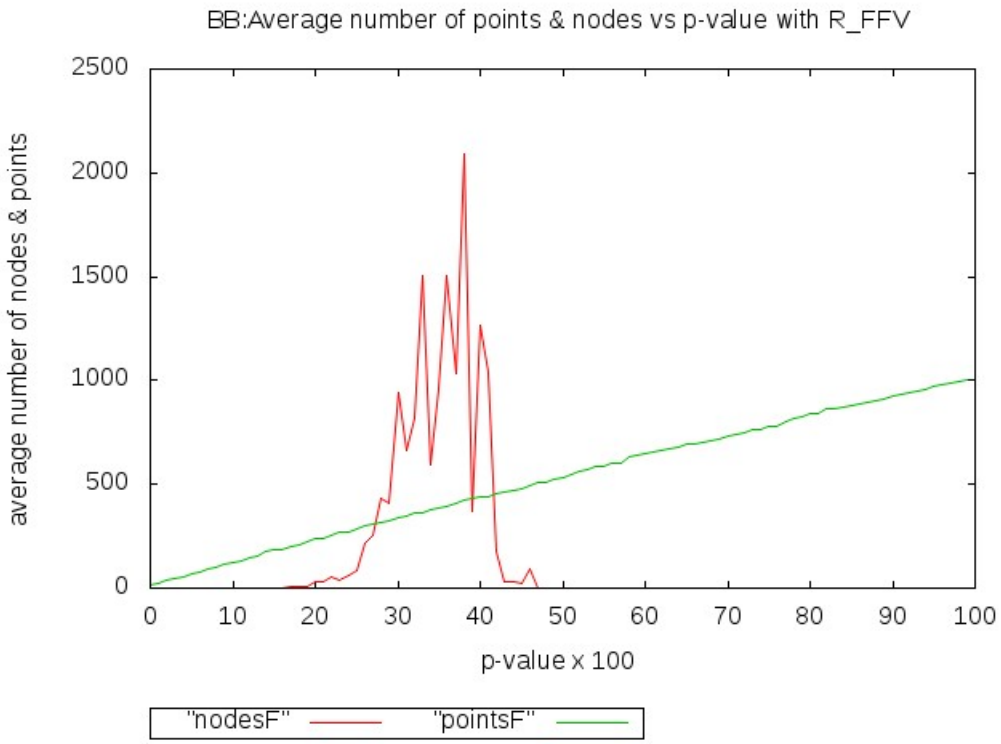

Figure 2.9: Nodes Explored to Cover Input Points with R_FFV

Following figure 2.10 graphically shows the running time :

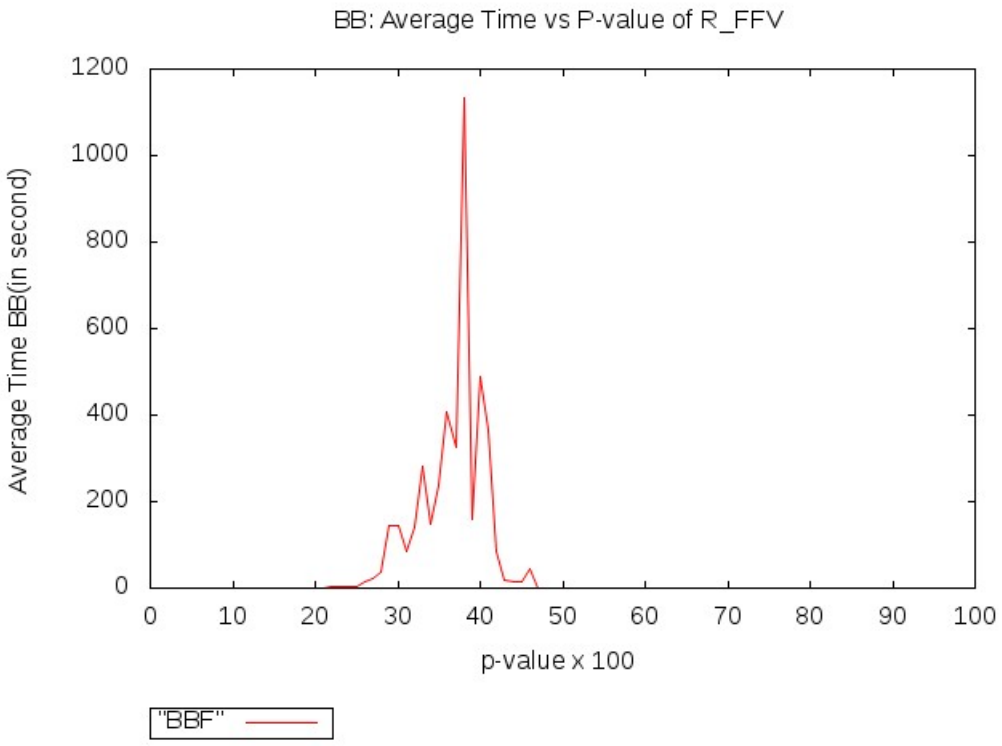

Figure 2.10: Running Time of Branch and Bounding Approach with R_FFV

Based on figure 2.10 we can say that, the running time of our branch and bound technique is much greater than all other running times. The highest time it takes to run the function is 1131.7172s at $p = 0.39$.

- Rounding the last fractional variable value (R_FFV) :

  Figure 2.11 contains the table of the experimental data. The first column of the table represents the $p - value$ and the second, third, fourth and fifth columns represent the average *avgBBLPL*, the *highestBBLPL*, the average *nodesL* and the average *pointsL* data respectively.

p-value	avgBBLPL	highestBBLPL	nodesL	pointsL
0.01	1	1	1	13
0.02	1	1	1	24.72
0.03	1	1	1	34.8
0.04	1	1	1	47.08
0.05	1	1	1	57.44
0.06	1	1	1	72.28
0.07	1	1	1	77.08
0.08	1	1	1	90.12
0.09	1	1	1	101.56
0.1	1.00036036	1.00900901	1.16	115.8
0.11	1.00036697	1.00917431	1.16	122
0.12	1.00071434	1.00900901	1.56	134.12
0.13	1.00096626	1.01020408	1.56	146.04
0.14	1.00197268	1.01149425	2.36	156.48
0.15	1.0018205	1.01587302	2.28	178.2
0.16	1.00330057	1.0212766	3.56	184.08
0.17	1.00254712	1.01428571	3.32	189.04
0.18	1.00350787	1.01408451	6.6	198.2
0.19	1.00421559	1.01408451	6.28	208
0.2	1.00499686	1.01570352	13.08	226.24
0.21	1.00838674	1.02248229	30.76	237.44
0.22	1.00902051	1.02208202	31.24	241.36
0.23	1.00875546	1.02087716	48.68	252.4
0.24	1.00994557	1.0375	46.92	272
0.25	1.01157047	1.02254283	53.32	273.36
0.26	1.01012172	1.02417189	114.6	284.08
0.27	1.01218073	1.02823759	114.04	299.2
0.28	1.01358244	1.0247733	504.76	305.6
0.29	1.01560492	1.02845023	245.96	315.96
0.3	1.01435654	1.02994419	293.48	325.76
0.31	1.01727804	1.0290505	563.72	337
0.32	1.01868755	1.03384062	2007.8	343.72
0.33	1.01781355	1.02817294	2240.92	359.6
0.34	1.01789349	1.03825137	511.72	366.08
0.35	1.01705099	1.02900122	1107.88	377.68
0.36	1.02034454	1.03739514	1102.92	385.6
0.37	1.01764566	1.03519757	1371.4	393.84
0.38	1.01726863	1.03299666	847	409.32
0.39	1.01478371	1.03206968	1562.2	428
0.4	1.01137819	1.02642388	436.68	430.24
0.41	1.01063626	1.02753839	944.36	436.44
0.42	1.01118458	1.03281939	646.92	437
0.43	1.0066359	1.02190557	59.16	452.84
0.44	1.00449564	1.02359332	188.52	462.6
0.45	1.00395065	1.02538071	18.76	469.92
0.46	1.00151072	1.0253999	644.84	481.52
0.47	1.00430235	1.03258145	71.64	493.2
0.48	1.00011508	1.00287711	1.32	509.2
0.49	1	1	1.24	512
0.5	1	1	1.16	523.16
0.51	1	1	1.08	533.88
0.52	1	1	1	544
0.53	1	1	1.08	562.16
0.54	1	1	1.08	567.8
0.55	1	1	1	583.96
0.56	1	1	1	589
0.57	1	1	1	599.12
0.58	1	1	1	600
0.59	1	1	1	633
0.6	1	1	1	638.04
0.61	1	1	1	651.44
0.62	1	1	1	653.64
0.63	1	1	1	663.48
0.64	1	1	1	671
0.65	1	1	1	678.32
0.66	1	1	1	691.6
0.67	1	1	1	693.76
0.68	1	1	1	698.68
0.69	1	1	1	706.64
0.7	1	1	1	716.08

p-value	avgBBLPL	highestBBLPL	nodesL	pointsL
0.71	1	1	1	731.2
0.72	1	1	1	740.8
0.73	1	1	1	751.28
0.74	1	1	1	761
0.75	1	1	1	764.2
0.76	1	1	1	778
0.77	1	1	1	782.32
0.78	1	1	1	800.64
0.79	1	1	1	814.36
0.8	1	1	1	822
0.81	1	1	1	837.68
0.82	1	1	1	839
0.83	1	1	1	861.48
0.84	1	1	1	865
0.85	1	1	1	868.72
0.86	1	1	1	876.96
0.87	1	1	1	886
0.88	1	1	1	894.88
0.89	1	1	1	904.08
0.9	1	1	1	911.8
0.91	1	1	1	923.56
0.92	1	1	1	930.68
0.93	1	1	1	941.28
0.94	1	1	1	950.96
0.95	1	1	1	959.32
0.96	1	1	1	971.64
0.97	1	1	1	979.16
0.98	1	1	1	988.44
0.99	1	1	1	996
1	1	1	1	1000

Figure 2.11: Computational Performance of Branch and Bound Algorithm with R_LFV

We graphically represent the experimental summary of the branch and bound algorithm with R_LFV in the following figure 2.12 :

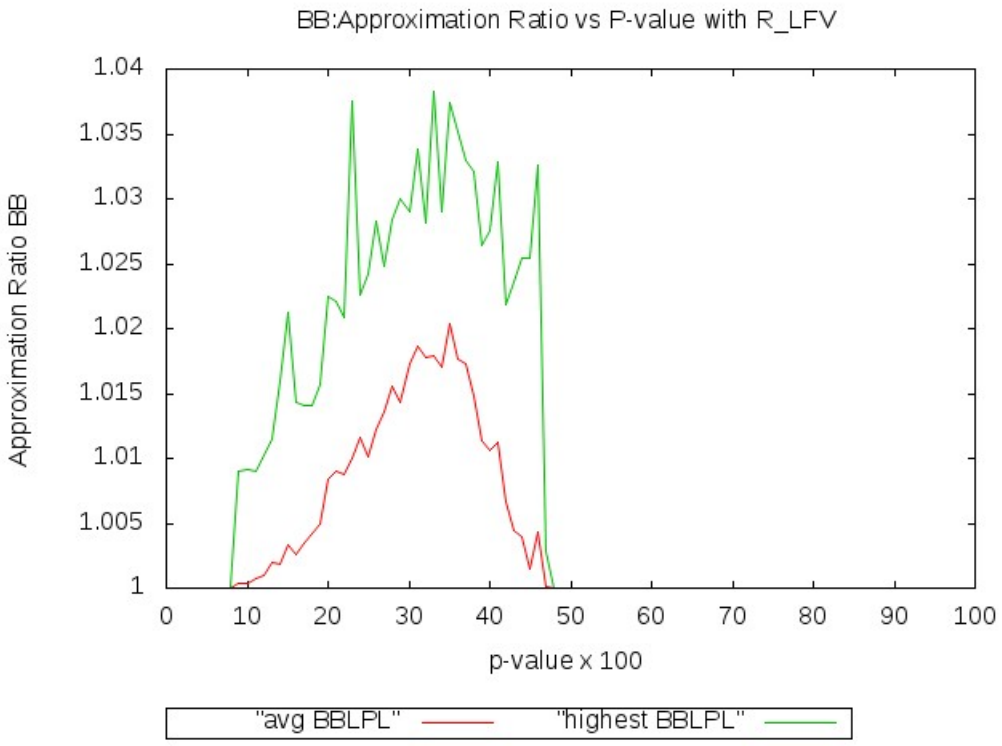

Figure 2.12: Graphical Representation of the Performance of Branch and Bound Algorithm with *R_LFV*

In figure 2.12 we plot the $p-values$ along the $x-axis$ and the approximation ratio(average of 25 values and highest among 25 values) along the $y-axis$. Based on figures 2.11 and 2.12 we can say that,

- Average approximation ratio of $\frac{BB\_OPT}{LP\_OPT}$:

  The approximation ratio starts increasing slowly for the input instance from $p = 0.1$ to $p = 0.36$. We obtain the highest approximation ratio at $p = 0.34$ which is 1.03825137. From $p = 0.37$, the approximation ratio starts decreasing slowly and it ends up at 1.

In figure 2.13 we plot the $p-values$ along the $x-axis$ and the *nodesL* and the *pointsL*(average of 25 values) along the $y-axis$. Based on figure 2.13 we can say that,

The *nodes* grows polynomially with the input instance starting from $p = 0.1$ to $p = 0.28$. We obtain the highest number of nodes explored at $p = 0.33$, which is 2240.92 and the average cardinality of the input instance is 360 . From $p = 0.39$, the number of nodes starts decreasing slowly and it ends up at 1.

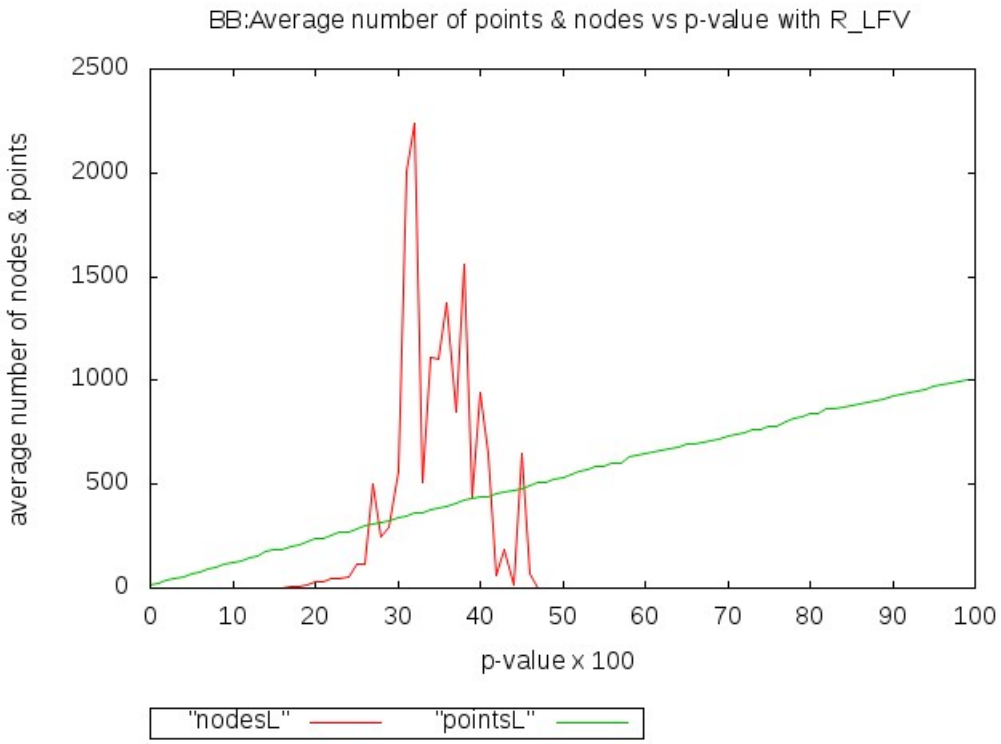

Figure 2.13: Nodes Explored to Cover Input Points with *R_LFV*

Following figure 2.14 shows the running time :

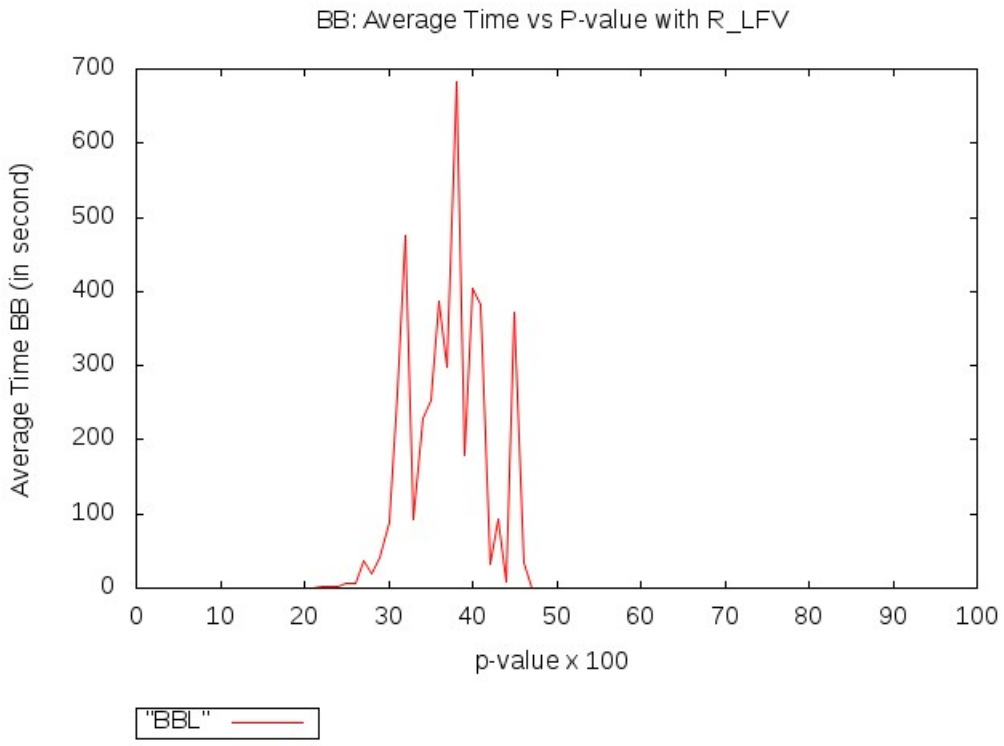

Figure 2.14: Running Time of Branch and Bounding Approach with *R_LFV*

Based on figure 2.14 we can say that, the running time of our branch and bound technique is much greater than all other running times. The highest time it takes to run the function is 683.2905s at $p = 0.39$.

- Rounding the largest fractional variable value (*R_LAFV*) :

  Figure 2.15 contains the table of the experimental data. The first column of the table represents the $p - value$ and the second, third, fourth and fifth columns represent the average *avgBBLPM*, the *highestBBLPM*, the average *nodesM* and the average *pointsM* data respectively.

p-value	avgBBLPM	highestBBLPM	nodesM	pointsM
0.01	1	1	1	13
0.02	1	1	1	24.72
0.03	1	1	1	34.8
0.04	1	1	1	47.08
0.05	1	1	1	57.44
0.06	1	1	1	72.28
0.07	1	1	1	77.08
0.08	1	1	1	90.12
0.09	1	1	1	101.56
0.1	1.00036036	1.00900901	1.16	115.8
0.11	1.00036697	1.00917431	1.16	122
0.12	1.00071434	1.00900901	1.56	134.12
0.13	1.00096626	1.01020408	1.56	146.04
0.14	1.00197268	1.01149425	2.36	156.48
0.15	1.0018205	1.01587302	2.6	178.2
0.16	1.00330057	1.0212766	3	184.08
0.17	1.00254712	1.01428571	3.96	189.04
0.18	1.00350787	1.01408451	6.76	198.2
0.19	1.00421559	1.01408451	6.92	208
0.2	1.00499686	1.01570352	8.84	226.24
0.21	1.00838674	1.02248229	34.76	237.44
0.22	1.00902051	1.02208202	40.76	241.36
0.23	1.00875546	1.02087716	43.08	252.4
0.24	1.00994557	1.0375	101.08	272
0.25	1.01157047	1.02254283	94.52	273.36
0.26	1.01012172	1.02417189	137.4	284.08
0.27	1.01218073	1.02823759	172.12	299.2
0.28	1.01358244	1.0247733	363.56	305.6
0.29	1.01560492	1.02845023	399.88	315.96
0.3	1.01435654	1.02994419	440.12	325.76
0.31	1.01727804	1.0290505	1057.56	337
0.32	1.01868755	1.03384062	1830.04	343.72
0.33	1.01781355	1.02817294	1683.16	359.6
0.34	1.01789349	1.03825137	1181.32	366.08
0.35	1.01705099	1.02900122	1126.6	377.68
0.36	1.02034454	1.03739514	2222.36	385.6
0.37	1.01764566	1.03519757	2834.76	393.84
0.38	1.01726863	1.03299666	1846.6	409.32
0.39	1.01478371	1.03206968	2793	428
0.4	1.01137819	1.02642388	328.36	430.24
0.41	1.01063626	1.02753839	1539.8	436.44
0.42	1.01118458	1.03281939	575.88	437
0.43	1.0066359	1.02190557	179	452.84
0.44	1.00449564	1.02359332	132.68	462.6
0.45	1.00395065	1.02538071	48.44	469.92
0.46	1.00151072	1.0253999	43.88	481.52
0.47	1.00430235	1.03258145	79	493.2
0.48	1.00011508	1.00287711	1.8	509.2
0.49	1	1	1.24	512
0.5	1	1	1.16	523.16
0.51	1	1	1.08	533.88
0.52	1	1	1	544
0.53	1	1	1.08	562.16
0.54	1	1	1.08	567.8
0.55	1	1	1	583.96
0.56	1	1	1	589
0.57	1	1	1	599.12
0.58	1	1	1	600
0.59	1	1	1	633
0.6	1	1	1	638.04
0.61	1	1	1	651.44
0.62	1	1	1	653.64
0.63	1	1	1	663.48
0.64	1	1	1	671
0.65	1	1	1	678.32
0.66	1	1	1	691.6
0.67	1	1	1	693.76
0.68	1	1	1	698.68
0.69	1	1	1	706.64
0.7	1	1	1	716.08

p-value	avgBBLPM	highestBBLPM	nodesM	pointsM
0.71	1	1	1	731.2
0.72	1	1	1	740.8
0.73	1	1	1	751.28
0.74	1	1	1	761
0.75	1	1	1	764.2
0.76	1	1	1	778
0.77	1	1	1	782.32
0.78	1	1	1	800.64
0.79	1	1	1	814.36
0.8	1	1	1	822
0.81	1	1	1	837.68
0.82	1	1	1	839
0.83	1	1	1	861.48
0.84	1	1	1	865
0.85	1	1	1	868.72
0.86	1	1	1	876.96
0.87	1	1	1	886
0.88	1	1	1	894.88
0.89	1	1	1	904.08
0.9	1	1	1	911.8
0.91	1	1	1	923.56
0.92	1	1	1	930.68
0.93	1	1	1	941.28
0.94	1	1	1	950.96
0.95	1	1	1	959.32
0.96	1	1	1	971.64
0.97	1	1	1	979.16
0.98	1	1	1	988.44
0.99	1	1	1	996
1	1	1	1	1000

Figure 2.15: Computational Performance of Branch and Bound Algorithm with *R_LAFV*

We graphically represent the experimental summary of the branch and bound algorithm with *R_LAFV* in the following figure 2.16 :

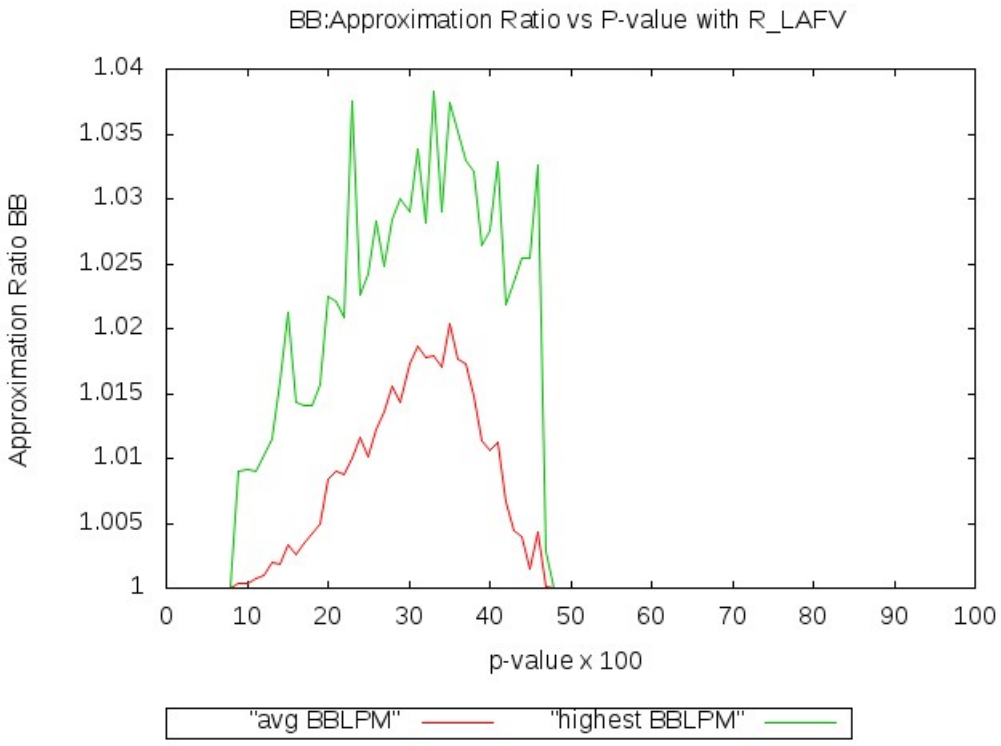

Figure 2.16: Graphical Representation of the Performance of Branch and Bound Algorithm with *R_LAFV*

In figure 2.16 we plot the $p-values$ along the $x-axis$ and the approximation ratio(average of 25 values and highest among 25 values) along the $y-axis$. Based on figures 2.15 and 2.16 we can say that,

- Average approximation ratio of $\frac{BB\_OPT}{LP\_OPT}$:

  The approximation ratio starts increasing slowly for the input instance from $p = 0.1$ to $p = 0.36$. We obtain the highest approximation ratio at $p = 0.34$ which is 1.03825137. From $p = 0.37$, the approximation ratio starts decreasing slowly and it ends up at 1.

In figure 2.17 we plot the $p-values$ along the $x-axis$ and the *nodesM* and the *pointsM*(average of 25 values) along the $y-axis$. Based on figure 2.17 we can say that,

The *nodes* grows polynomially with the input instance starting from $p = 0.1$ to $p = 0.33$. We obtain the highest number of nodes explored at $p = 0.37$, which is 2834.75 and the average cardinality of the input instance is 393.84 . From $p = 0.41$, the number of nodes starts decreasing slowly and it ends up at 1.

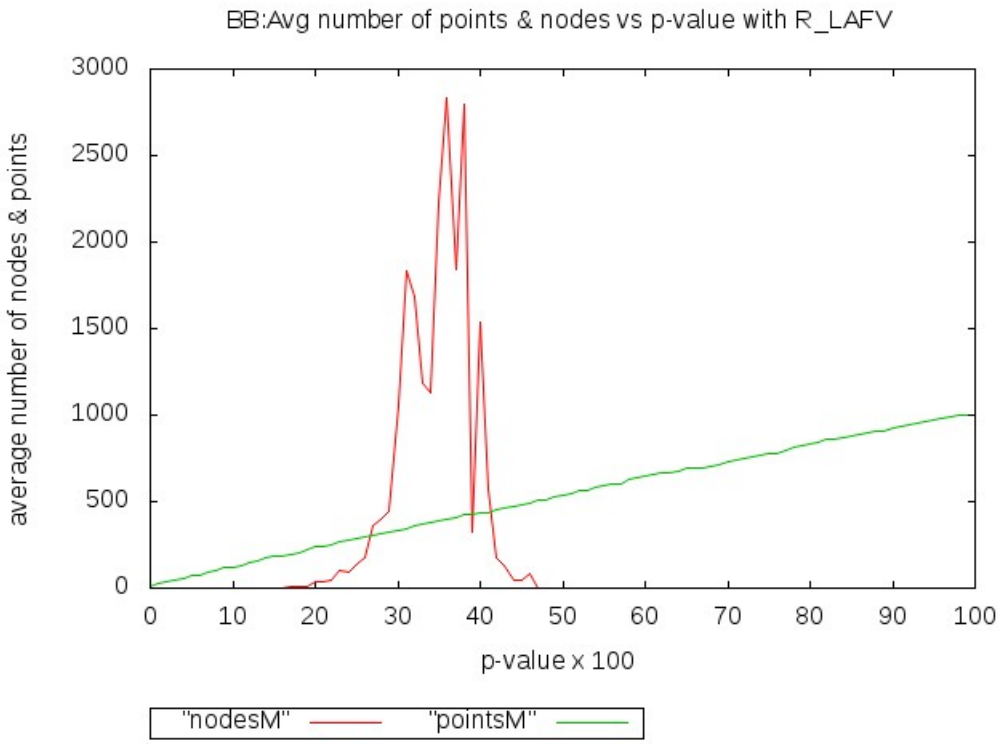

Figure 2.17: Nodes Explored to Cover Input Points with *R_LAFV*

Following figure 2.18 shows the running time :

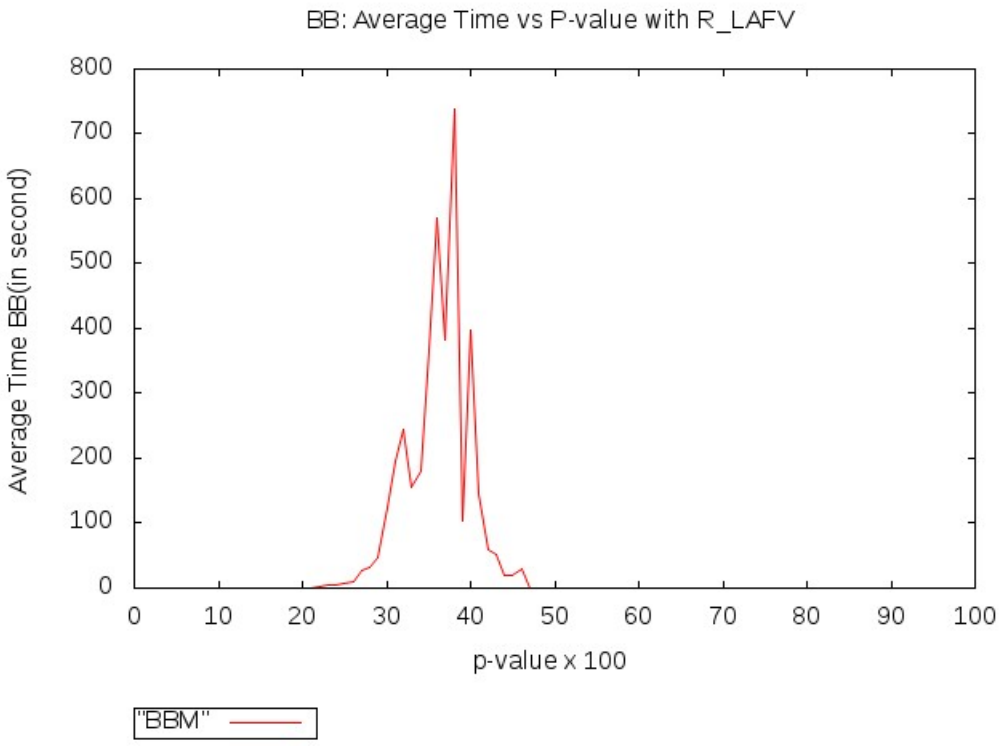

Figure 2.18: Running Time of Branch and Bounding Approach with *R_LAFV*

Based on figure 2.18 we can say that, the running time of our branch and bound technique is much greater than all other running times. The highest time it takes to run the function is 738.040019s at $p = 0.39$.

As Branch and bound approach runs in exponential time, so we tried to find out a way which takes less running time. We could come to the conclusion that, the approach of rounding the last fractional variable value (*R_LFV*) to 1 at each recursive call takes the minimum time compared to the other two approaches. Because this approach explores the minimum number of nodes to cover all the points.

Chapter 3

Iterative Rounding Algorithm 2

3.1 Introduction

In this chapter, we discuss another scheme of iterative rounding method [8]. In Section 3.2 we discuss the iterative rounding Algorithm 2 that we implemented. In Section 3.3 the approximation ratio of the algorithm is discussed. Finally in Section 3.4 we present the experimental results for the iterative rounding approach.

3.2 Iterative Rounding Algorithm 2

In this algorithm, the concept of iterative rounding and the primal-dual method has been used. Let us consider α is the approximation factor of the algorithm, ε is a constant with value $0 < \varepsilon \leq 1$ and d is the dimension (3 for the experiments). The algorithm is as follows:

1. At first, the algorithm solves the primal *LP*. The solution returns the minimum number of axis parallel lines to cover all the points and the optimal values to the decision variables fractionally.

2. In the next step, the algorithm solves the dual of the primal *LP*.

3. If the primal *LP* solution LP_OPTP returns an integral output where all the variables have integral values, then that is the required result.

4. In the case of the dual *LP* solution $LP\_OPTD \geq \frac{n}{\alpha}$ where n is the number of points in the subproblem, the algorithm returns any cover to the point cover problem.

5. In the case of the dual *LP* solution $LP\_OPTD \leq \frac{d}{\varepsilon}$, the algorithm returns the cheapest cover by enumerating all the subsets of lines which have size no more than $\frac{d^2}{\varepsilon}$.

6. If there is a fractional value $\geq \frac{1}{\alpha}$, then the algorithm rounds it up to 1 and solves the subproblem recursively .

7. If none of the above condition executes, the algorithm selects a point p with the dual variable value $\leq \frac{1}{\alpha}$, then the algorithm includes all the lines passing through that point in the cover and solves the subproblem instance recursively. Such a point p always exists, otherwise if the dual variable value would be $> \frac{1}{\alpha}$ then for all the n points the algorithm would get a dual solution $> \frac{n}{\alpha}$ which follows step 4.

8. The recursive calls keep following steps 3, 4, 5, 6 and 7 until the integral solution is obtained.

Here is the Algorithm 2 of iterative rounding method :

Algorithm 4 Iterative Rounding Algorithm 2

Require: A minimization LP in standard form.

Ensure: Integral solution, IR2 to the LP.

1: **procedure** IR2_func*(min LP_OPT, $Afv \geq 1, fv \geq 0$)*
2: *find the optimal primal LP solution LP_OPTP.*
3: *find the optimal dual LP solution LP_OPTD.*
4: **if** *no fractional variable fv_i exists* **then**
5: ***return*** *LP_OPTP.*
6: **end if**
7: **if** *LP_OPTD* $\geq \frac{n}{\alpha}$ **then**
8: ***return*** *any cover of the point cover problem.*
9: **end if**
10: **if** *LP_OPTD* $\leq \frac{d}{\varepsilon}$ **then**
11: ***return*** *the cheapest cover by enumerating all the subsets of lines which has size no more than* $\frac{d^2}{\varepsilon}$.
12: **end if**
13: **if** *a fractional variable $fv_i \geq \frac{1}{\alpha}$ exists* **then**
14: **comment:** round that variable to 1.
15: ***return*** IR2_func*(min LP_OPT, $Afv \geq 1$, $fv_i = 1, fv \geq 0$)*.
16: **end if**
17: **if** *a point p with dual variable $dv_i \leq \frac{1}{\alpha}$ exists* **then**
18: **comment:** round all the d lines L_p going through that point p to 1.
19: ***return*** IR2_func*(min LP_OPT, $Afv \geq 1$, $fv_i = 1, \forall i \in L_p, fv \geq 0$)*.
20: **end if**
21: **end procedure**

A subset of variables is rounded at each recursive call of the algorithm. The current instance of the problem is updated by reducing its size based on the rounded variables and

this updated instance is passed to the next recursive call.

3.3 Approximation Ratio

Conjecture 3.1. *The point cover problem can be approximated within a factor of* $\frac{1}{4} + \sqrt{\frac{1}{4} + d(1-\varepsilon)}$ *within polynomial time for constants d and* $0 < \varepsilon \leq 1$.

The iterative rounding Algorithm 2 does not rely on linear independence. The algorithm shows that either there is a large fractional value in the primal solution or there is a set of dual variables to which the cost of the lines in any given iteration can be charged. In the algorithm it is assumed that, the number of points is n. Later α is chosen to minimize the performance ratio of the algorithm. For a few cases of the algorithm an α-approximate solution can be constructed in the current call. If none of the above cases (lines 4-12) are met then the algorithm makes a recursive call in the following way. If a fractional primal variable $fv_i \geq \frac{1}{\alpha}$ exists then line i is picked in the cover and the algorithm recursively computes the cover for the remaining points not covered by line i. If no such primal variable exists then a point p with dual variable $dv_i \leq \frac{1}{\alpha}$ exists. All the lines passing through point p are picked in the cover and the supproblem is solved recursively. The iterative rounding Algorithm 2 takes a linear program for an instance of the point cover problem as input and returns a set of lines as an integral vector. The function call can return on lines 5, 8, 11, 15, 19. The return on lines 5, 8, 11 are the base cases [8].

3.4 Experiments and Results

In this section, we present the empirical data obtained by running experiments on the iterative rounding Algorithm 2 which computes the approximation ratio based on a 10x10x10 grid. We used a range of p values from 0.01 to 1 with a 0.01 difference. Each grid point was picked with probability p in the input instance. We repeat each experiment 25 times for each $p - value$ and took the average of 25 values.

We used Octave 3.8.1 for the implementation of the algorithms. We used the GLPK solver for computing the *LP* value and the *IP* value for each input instance. All the experiments presented here were conducted on a 2.10 GHz processor with 8 GB of RAM in Ubuntu 14.04 environment.

3.4.1 Experimental Results of the Second Iterative Rounding Approach

Here we present the experimental results of iterative rounding Algorithm 2 :

- *avgIR*2*LP* represents the average approximation ratio of the integral solution *IR*2 and the optimal *LP* solution *LP_OPT*.
- *avgIR*2*IP* represents the average approximation ratio of the integral solution *IR*2 and the optimal *IP* solution *IP_OPT*.
- *highestIR*2*LP* represents the highest approximation ratio of the integral solution *IR*2 and the optimal *LP* solution *LP_OPT*.
- *highestIR*2*IP* represents the highest approximation ratio of the integral solution *IR*2 and the optimal *IP* solution *IP_OPT*.

Figure 3.1 summarizes the results. The first column of the table represents the $p-value$ and second, third, fourth and fifth columns represent the average *avgIR*2*LP*, the average *avgIR*2*IP*, the *highestIR*2*LP* and the *highestIR*2*IP* data respectively.

p-value	avgIR2LP	avgIR2IP	highestIR2LP	highestIR2IP
0.01	1	1	1	1
0.02	1	1	1	1
0.03	1	1	1	1
0.04	1	1	1	1
0.05	1	1	1	1
0.06	1	1	1	1
0.07	1	1	1	1
0.08	1	1	1	1
0.09	1	1	1	1
0.1	1.00471868	1.00432615	1.0990991	1.08928571
0.11	1.00989612	1.00948244	1.13761468	1.12727273
0.12	1.03026028	1.02939172	1.42342342	1.41071429
0.13	1.0213198	1.02032997	1.13114754	1.13114754
0.14	1.03422252	1.0320972	1.24786325	1.23728814
0.15	1.02934848	1.02807833	1.19444444	1.19444444
0.16	1.03547363	1.03260657	1.13846154	1.13846154
0.17	1.0347958	1.03208411	1.20567376	1.1971831
0.18	1.05552962	1.05184249	1.2033543	1.2
0.19	1.06555698	1.06154114	1.35576923	1.34285714
0.2	1.06852791	1.06381072	1.17100372	1.15384615
0.21	1.05124533	1.04398446	1.21799308	1.2
0.22	1.06039769	1.05355462	1.25239617	1.24050633
0.23	1.07309466	1.06600038	1.25316456	1.25316456
0.24	1.06958027	1.06049923	1.225	1.20987654
0.25	1.06954414	1.05878371	1.17857143	1.16470588
0.26	1.05949634	1.05126069	1.19492158	1.17647059
0.27	1.06639112	1.05691291	1.2360515	1.22727273
0.28	1.08279068	1.06925933	1.16395586	1.14772727
0.29	1.09379304	1.08043225	1.31069721	1.28735632
0.3	1.07513205	1.06084718	1.19638826	1.19101124
0.31	1.05149447	1.03815921	1.18800312	1.17647059
0.32	1.08079898	1.06456647	1.28603104	1.26086957
0.33	1.07962248	1.06255687	1.21771218	1.19565217
0.34	1.07395372	1.0595499	1.226663	1.20430108
0.35	1.1051579	1.08944259	1.23957259	1.22340426
0.36	1.07472481	1.05819523	1.22207411	1.21052632
0.37	1.06341819	1.04927925	1.14399119	1.12631579
0.38	1.05724019	1.04566694	1.14911315	1.1443299
0.39	1.04662251	1.03510662	1.14148958	1.12371134
0.4	1.04601893	1.03847724	1.1866531	1.18367347
0.41	1.04628359	1.03861629	1.16493541	1.1443299
0.42	1.02365284	1.01728985	1.05998241	1.05208333
0.43	1.04561011	1.04161493	1.22019104	1.21
0.44	1.03343104	1.0304384	1.16526904	1.15
0.45	1.03755372	1.03552373	1.21827411	1.21212121
0.46	1.01810577	1.01738677	1.06060606	1.06060606
0.47	1.01637915	1.01487289	1.11	1.11
0.48	1.01936817	1.01924848	1.08	1.08
0.49	1.02132525	1.02132525	1.1010101	1.1010101
0.5	1.01282424	1.01282424	1.08	1.08
0.51	1.0216569	1.0216569	1.14	1.14
0.52	1.01844881	1.01844881	1.08	1.08
0.53	1.0168	1.0168	1.08	1.08
0.54	1.01731363	1.01731363	1.1010101	1.1010101
0.55	1.02808081	1.02808081	1.12121212	1.12121212
0.56	1.01120808	1.01120808	1.04	1.04
0.57	1.0184404	1.0184404	1.06060606	1.06060606
0.58	1.0104	1.0104	1.04	1.04
0.59	1.0176	1.0176	1.08	1.08
0.6	1.012	1.012	1.06	1.06
0.61	1.0088	1.0088	1.04	1.04
0.62	1.0096	1.0096	1.06	1.06
0.63	1.0168	1.0168	1.08	1.08
0.64	1.02	1.02	1.08	1.08
0.65	1.0096	1.0096	1.06	1.06
0.66	1.0088	1.0088	1.08	1.08
0.67	1.01840808	1.01840808	1.08	1.08
0.68	1.0064	1.0064	1.06	1.06
0.69	1.0096	1.0096	1.08	1.08
0.7	1.0088	1.0088	1.06	1.06

p-value	avgIR2LP	avgIR2IP	highestIR2LP	highestIR2IP
0.71	1.0128	1.0128	1.06	1.06
0.72	1.004	1.004	1.04	1.04
0.73	1.0112	1.0112	1.04	1.04
0.74	1.0048	1.0048	1.04	1.04
0.75	1.0064	1.0064	1.04	1.04
0.76	1.0064	1.0064	1.04	1.04
0.77	1.008	1.008	1.04	1.04
0.78	1.0104	1.0104	1.06	1.06
0.79	1.0136	1.0136	1.04	1.04
0.8	1.008	1.008	1.04	1.04
0.81	1.0088	1.0088	1.06	1.06
0.82	1.0112	1.0112	1.06	1.06
0.83	1.0072	1.0072	1.04	1.04
0.84	1.0072	1.0072	1.04	1.04
0.85	1.008	1.008	1.06	1.06
0.86	1.0048	1.0048	1.04	1.04
0.87	1.0056	1.0056	1.04	1.04
0.88	1.0088	1.0088	1.06	1.06
0.89	1.0056	1.0056	1.08	1.08
0.9	1.0048	1.0048	1.02	1.02
0.91	1.012	1.012	1.1	1.1
0.92	1.0056	1.0056	1.04	1.04
0.93	1.0048	1.0048	1.04	1.04
0.94	1.0056	1.0056	1.06	1.06
0.95	1.008	1.008	1.08	1.08
0.96	1.0048	1.0048	1.04	1.04
0.97	1.0048	1.0048	1.02	1.02
0.98	1.0048	1.0048	1.02	1.02
0.99	1.0064	1.0064	1.04	1.04
1	1	1	1	1

Figure 3.1: Computational Performance of Iterative Rounding Algorithm 2

We graphically represent the experimental summary of iterative rounding Algorithm 2 in the following figure 3.2 :

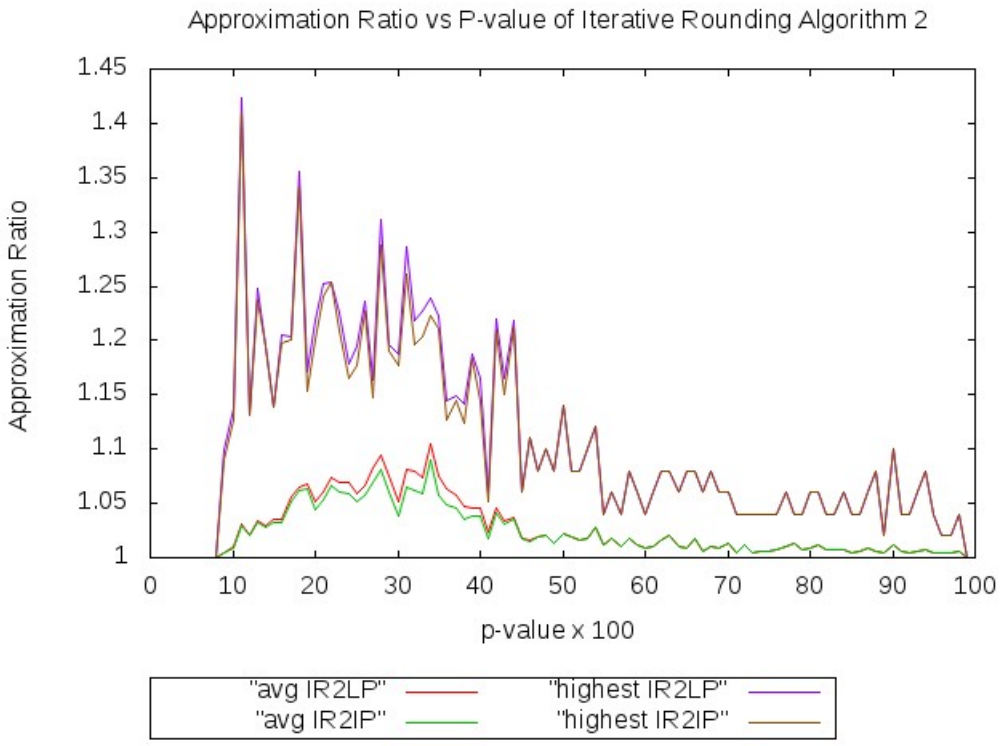

Figure 3.2: Graphical Representation of the Performance of Iterative Rounding Algorithm 2

In figure 3.2 we plot the $p-values$ along the $x-axis$ and the approximation ratio(average of 25 values and highest among 25 values) along the $y-axis$. Based on figures 3.1 and 3.2 we can say that,

- Average approximation ratio of $\frac{IR2}{LP\_OPT}$:

  In the case of the input instance of the $p-value$ from 0.01 to 0.09, we receive integral LP solutions. So the approximation ratios are 1. For the instances starting from $p-value$ 0.1 to 0.35, the approximation ratio increases slowly. After that, from $p = 0.36$ to $p = 1$, the approximation ratios of the respective instances decrease slowly and end up at 1 when $p = 1$. At $p = 0.12$, we obtain the highest approximation ratio which is 1.423423.

- Approximation ratio of $\frac{IR2}{IP\_OPT}$:

The approximation ratio starts increasing slowly for the input instance from $p = 0.11$ to $p = 0.35$. From $p = 0.36$, the approximation ratio starts decreasing slowly and it ends up at 1 when $p = 1$. We obtain the highest approximation ratio at $p = 0.12$, which is 1.410714.

In figure 3.3 we compare the running time of the GLPK *LP* solver, the GLPK *IP* solver and the iterative rounding approach 2 based on a 10x10x10 data grid. We plot the average time of 25 results for each $p - value$. All the times are computed in seconds.

- *LP* represents the average time to compute LP_OPT value.
- *IP* represents the average time to compute IP_OPT value.
- *IR*2 represents the average time to compute *IR*2 value.

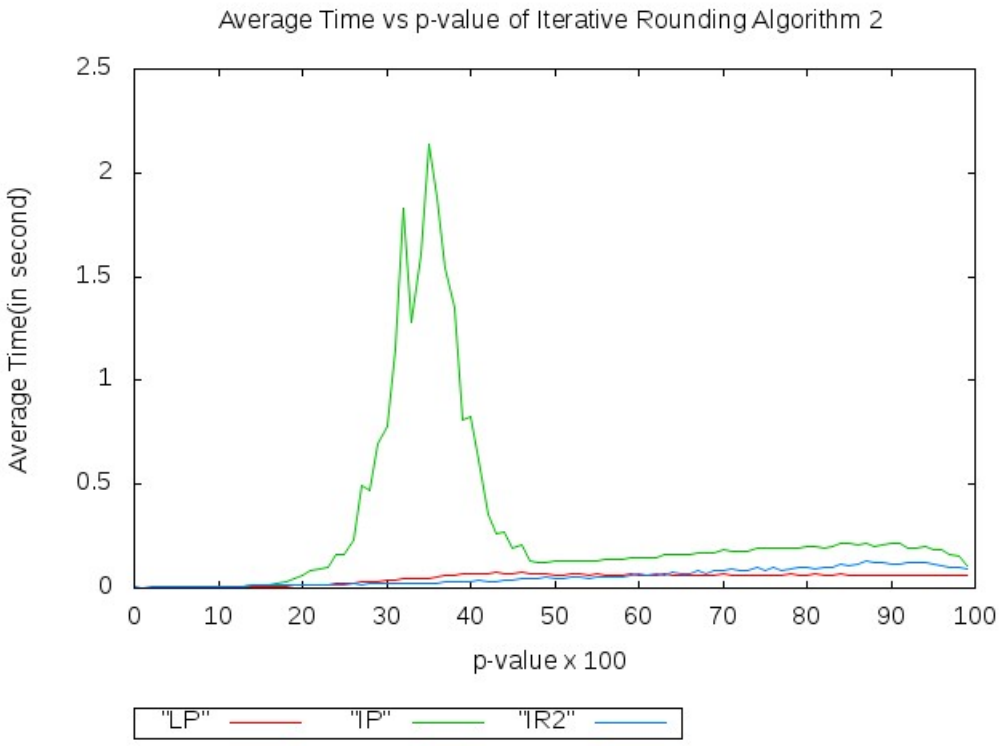

Figure 3.3: Running Time of GLPK LP Solver, GLPK IP Solver and Iterative Rounding Approach 2

Based on figure 3.3 we can say that, for the input instance from $p = 0.18$ to $p = 0.48$, the iterative rounding Algorithm 2 shows significant improvement in the running time compared to the running time of the GLPK *IP* solver. So we can conclude that the iterative rounding approach performs the best in the case of the running time.

Chapter 4

Conclusion

4.1 Introduction

In this chapter, we summarize the experimental results of the iterative rounding Algorithm 1, iterative rounding Algorithm 2 and branch and bound algorithm and compare the approximation ratios with the best known one in Section 4.2. we discuss the comparative analysis between the two iterative rounding approaches in Section 4.3. Finally in Section 4.4 we present significance of our branch and bound approach.

4.2 Summary

This chapter is a summary of the results and conclusions. In this thesis, we consider the problem of covering points by axis parallel lines in R^d. We presented an iterative rounding approach to compute an integral solution. We designed a branch and bound approach to compute the optimal integral solution. We also implemented another $\alpha-$approximation iterative rounding approach [8].

The current best known approximation ratio for covering points with axis parallel lines in d dimension is $\frac{d}{2}$ [8].

According to our experiments ($d = 3$) :

- best known approximation ratio $= \frac{3}{2} = 1.5$.
- Approximation ratios of iterative rounding Algorithm 1 :
  - $\frac{IR1}{IP}$ is 1.1041667.

  - $\frac{IR1}{LP}$ is 1.1272219.

- Approximation ratios of iterative rounding Algorithm 2 :
  - $\frac{IR2}{IP}$ is 1.410714.
  - $\frac{IR2}{LP}$ is 1.423423.

- For branch and bound algorithm :
  - Approximation ratio, $\frac{BB}{IP}$ is 1.
  - Integrality gap, $\frac{BB}{LP}$ is 1.038251.

from the above results we can come to the conclusion that, both of the iterative rounding algorithms show better approximation ratio compared to the best known one.

4.3 Comparison Between the Two Iterative Rounding Approaches

In this Section, we present a comparative analysis of the two iterative rounding approaches we have mentioned earlier in Chapter 2 and Chapter 3 respectively.

In the case of the iterative rounding Algorithm 1, we round the largest fractional variable value to 1 in each iteration until all the variables are integral.

Iterative rounding Algorithm 2 uses the primal dual approach and it solves the problem recursively. it uses a new idea. If any primal fractional variable with value $\geq \frac{1}{\alpha}$ exists, then that variable is rounded to 1 in the recursive call. Otherwise, if any dual variable with value $\leq \frac{1}{\alpha}$ exists, then the variable values for all the lines going through that point are rounded to 1. For $d = 3$ the algorithm rounds three variables to 1 in a single recursive call.

We used the same set of the input instance for both of the algorithms. Figure 4.1 summarizes the results. The first column of the table represents the $p - value$ and the second, third, fourth and fifth columns represent the $highestIR1LP$, the $highestIR2LP$, the $highestIR1IP$ and the $highestIR2IP1$ data respectively.

p-value	highestIR1LP	highestIR2LP	highestIR1IP	highestIR2IP
0.01	1	1	1	1
0.02	1	1	1	1
0.03	1	1	1	1
0.04	1	1	1	1
0.05	1	1	1	1
0.06	1	1	1	1
0.07	1	1	1	1
0.08	1	1	1	1
0.09	1	1	1	1
0.1	1.00900901	1.0990991	1	1.08928571
0.11	1.00917431	1.13761468	1	1.12727273
0.12	1.00900901	1.42342342	1	1.41071429
0.13	1.01020408	1.13114754	1	1.13114754
0.14	1.01639344	1.24786325	1.01639344	1.23728814
0.15	1.01587302	1.19444444	1	1.19444444
0.16	1.0212766	1.13846154	1.01492537	1.13846154
0.17	1.02857143	1.20567376	1.01408451	1.1971831
0.18	1.01860465	1.2033543	1.01408451	1.2
0.19	1.02552719	1.35576923	1.01428571	1.34285714
0.2	1.04089219	1.17100372	1.02631579	1.15384615
0.21	1.04207851	1.21799308	1.03797468	1.2
0.22	1.04958047	1.25239617	1.03797468	1.24050633
0.23	1.05521472	1.25316456	1.04878049	1.25316456
0.24	1.0625	1.225	1.025	1.20987654
0.25	1.06097561	1.17857143	1.04819277	1.16470588
0.26	1.03958178	1.19492158	1.02439024	1.17647059
0.27	1.06329114	1.2360515	1.03703704	1.22727273
0.28	1.07188931	1.16395586	1.04597701	1.14772727
0.29	1.06996985	1.31069721	1.04444444	1.28735632
0.3	1.06281475	1.19638826	1.03448276	1.19101124
0.31	1.07571931	1.18800312	1.05494505	1.17647059
0.32	1.09584971	1.28603104	1.06521739	1.26086957
0.33	1.08780622	1.21771218	1.06521739	1.19565217
0.34	1.09522196	1.226663	1.07446809	1.20430108
0.35	1.09286169	1.23957259	1.07291667	1.22340426
0.36	1.09447754	1.22207411	1.08163265	1.21052632
0.37	1.1040573	1.14399119	1.08163265	1.12631579
0.38	1.11043052	1.14911315	1.09090909	1.1443299
0.39	1.11381778	1.14148958	1.09	1.12371134
0.4	1.10859179	1.1866531	1.09	1.18367347
0.41	1.12722192	1.16493541	1.10416667	1.1443299
0.42	1.09287617	1.05998241	1.08	1.05208333
0.43	1.08986269	1.22019104	1.07070707	1.21
0.44	1.12346985	1.16526904	1.1010101	1.15
0.45	1.0421168	1.21827411	1.04	1.21212121
0.46	1.0756646	1.06060606	1.04901961	1.06060606
0.47	1.07152229	1.11	1.06060606	1.11
0.48	1.00287711	1.08	1	1.08
0.49	1.01010101	1.1010101	1.01010101	1.1010101
0.5	1.01010101	1.08	1.01010101	1.08
0.51	1	1.14	1	1.14
0.52	1	1.08	1	1.08
0.53	1.01	1.08	1.01	1.08
0.54	1	1.1010101	1	1.1010101
0.55	1	1.12121212	1	1.12121212
0.56	1	1.04	1	1.04
0.57	1	1.06060606	1	1.06060606
0.58	1	1.04	1	1.04
0.59	1	1.08	1	1.08
0.6	1	1.06	1	1.06
0.61	1	1.04	1	1.04
0.62	1	1.06	1	1.06
0.63	1	1.08	1	1.08
0.64	1	1.08	1	1.08
0.65	1	1.06	1	1.06
0.66	1	1.08	1	1.08
0.67	1	1.08	1	1.08
0.68	1	1.06	1	1.06
0.69	1	1.08	1	1.08
0.7	1	1.06	1	1.06

p-value	highestIR1LP	highestIR2LP	highestIR1IP	highestIR2IP
0.71	1	1.06	1	1.06
0.72	1	1.04	1	1.04
0.73	1	1.04	1	1.04
0.74	1	1.04	1	1.04
0.75	1	1.04	1	1.04
0.76	1	1.04	1	1.04
0.77	1	1.04	1	1.04
0.78	1	1.06	1	1.06
0.79	1	1.04	1	1.04
0.8	1	1.04	1	1.04
0.81	1	1.06	1	1.06
0.82	1	1.06	1	1.06
0.83	1	1.04	1	1.04
0.84	1	1.04	1	1.04
0.85	1	1.06	1	1.06
0.86	1	1.04	1	1.04
0.87	1	1.04	1	1.04
0.88	1	1.06	1	1.06
0.89	1	1.08	1	1.08
0.9	1	1.02	1	1.02
0.91	1	1.1	1	1.1
0.92	1	1.04	1	1.04
0.93	1	1.04	1	1.04
0.94	1	1.06	1	1.06
0.95	1	1.08	1	1.08
0.96	1	1.04	1	1.04
0.97	1	1.02	1	1.02
0.98	1	1.02	1	1.02
0.99	1	1.04	1	1.04
1	1	1	1	1

Figure 4.1: Comparative Computational Performance of Iterative Rounding Approaches

We graphically represent the experimental summary in the following figure 4.2 :

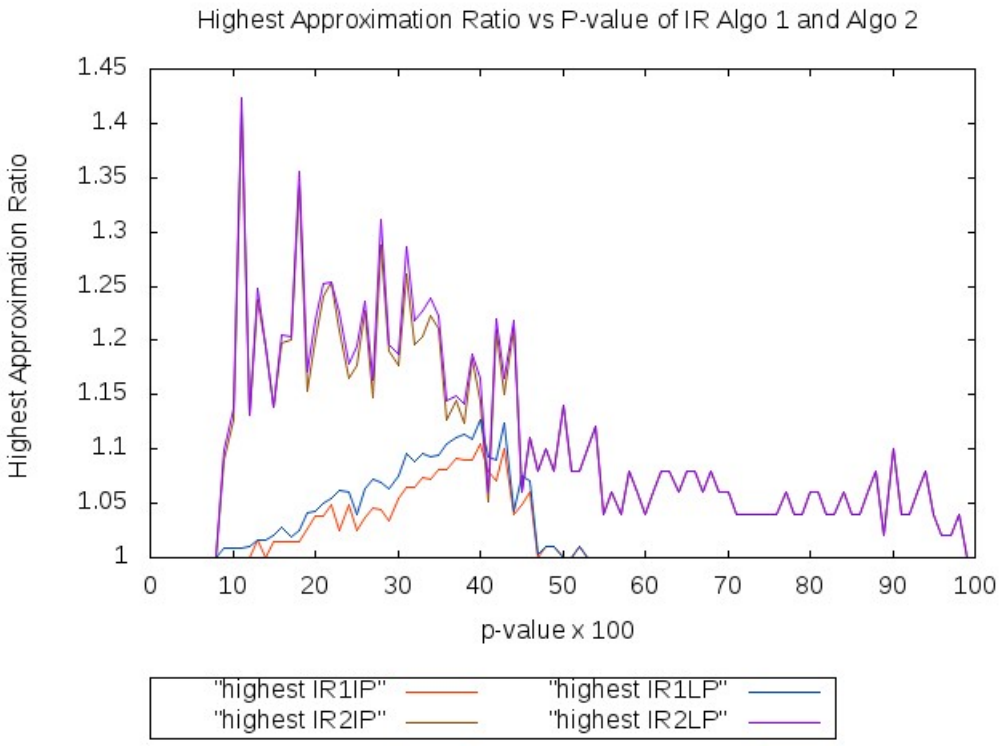

Figure 4.2: Graphical Representation of Comparative Performance of Iterative Rounding Approaches

In figure 4.2, we plot the $p-values$ along the $x-axis$ and the approximation ratio (highest among 25 values) along the $y-axis$. Based on figures 4.1 and 4.2 we can say that, the highest approximation ratio $\frac{IR1}{LP}$ is 1.1272219 and $\frac{IR2}{LP}$ is 1.423423. Similarly The highest approximation ratio $\frac{IR1}{IP}$ is 1.1041667 and $\frac{IR2}{IP}$ is 1.410714. So we can conclude that, in case of the resulting data Algorithm 1 performs better than Algorithm 2.

Figure 4.3 compares the running time of the iterative rounding functions based on a 10x10x10 data grid for both of the algorithms. We plot the $p-values$ along $x-axis$ and the average running time along $y-axis$. All the times are computed in seconds.

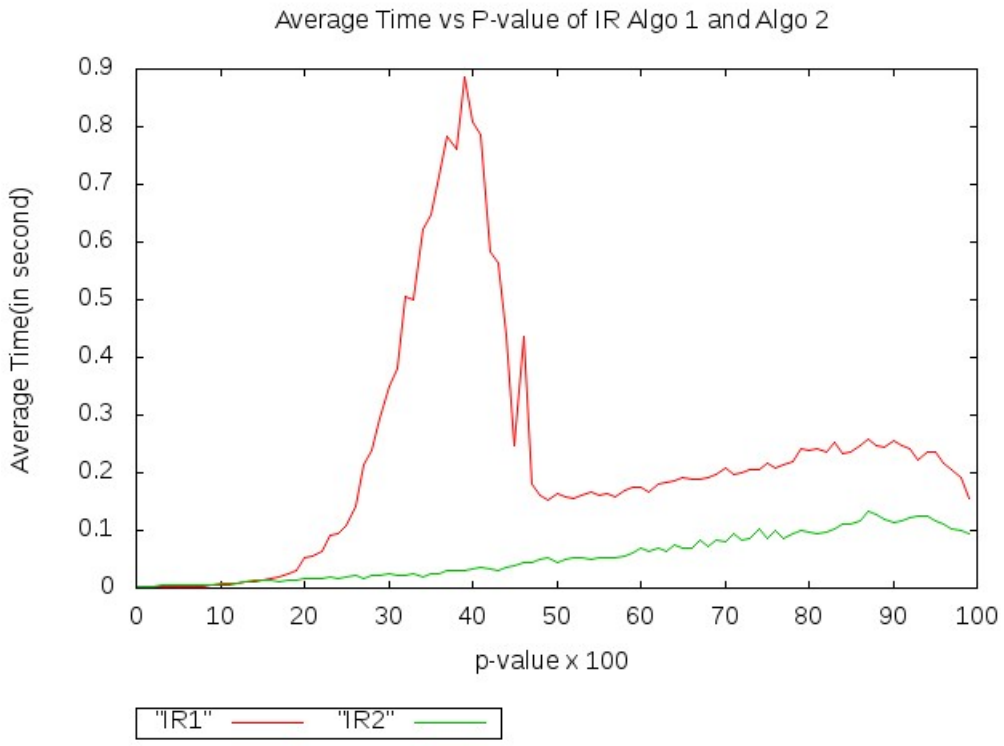

Figure 4.3: Comparative Running Time of Iterative Rounding Approaches

According to figure 4.3, the iterative function of Algorithm 1 requires more time compared to the Algorithm 2. So in case of running time Algorithm 2 performs better than Algorithm 1.

In case of resulting data and running time the performance of the algorithms differ. The reason is that, the Algorithm 1 rounds only one fractional variable value to 1, so picks only one line in the solution in each iteration. But in case of the Algorithm 2, there are some base cases which may return integral solution in a single call. Moreover there is a case when Algorithm 2 rounds d fractional variable values to 1 when there exists a dual variable with value $\leq \frac{1}{\alpha}$, so picks all the d lines passing through that point in the cover in a single recursive call.

From the above discussion we can conclude that, our Algorithm 1 shows better performance in terms of the value of the integral solution but shows worse performance in the case of

the running time as compared to the Algorithm 2. But the drawback of our Algorithm 1 is that, we could not be able to find any approximation factor for the Algorithm 1 but the Algorithm 2 offers an $\alpha-$approximation factor where $\alpha = \frac{1}{4} + \sqrt{\frac{1}{4} + d(1-\varepsilon)}$.

4.4 Significance of the Branch and Bound Approach

We are the first who have used the branch and bound technique to solve the problem of covering points by axis parallel lines. We have obtained the optimal integral solution of the above mentioned problem from our branch and bound algorithm. For some large input instances, the GLPK *IP* solver cannot return any output. But our branch and bound algorithm computes the optimal outputs for all those instances.

The only drawback of our algorithm is that the running time can be worse depending on the size of the input instance. In order to reduce this problem a bit, we have applied three strategies in case of picking the fractional variable which are the first one in the ordering, the last one in the ordering and the largest one in the ordering. As all the above techniques return the same result for the experimental data, based on the running time we can conclude that, rounding the last fractional variable value to 1 in each recursive call takes the least time to compute the solution. So for better performance we can use the technique of rounding the last fractional variable while using the branch and bound algorithm.

Bibliography

[1] George B. Dantzig. Origins of the simplex method. *TECHNICAL REPORT SOL 87-5*, May 1987.

[2] Daya Ram Gaur and Binay Bhattacharya. Covering points by axis parallel lines. *Proceedings of the 23rd European Workshop on Computational Geometry*, March 19-21, 2007.

[3] R. Hassin and N.Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29–42, 1991.

[4] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982.

[5] K. Jain. A factor 2 approximation algorithms for the generalized steiner network problem. *Combinatorica 21, 39-60*, 2001.

[6] Narendra Karmakar. A new polynomial time approximation algorithms for linear programming. *Combinatorica*, (4):373–395, 1984.

[7] V. Klee and G. J. Minty. How good is the simplex method. *Inequalities III :159-172*, 1972.

[8] Daya Ram Gaur, Robert Benkoczi, Kawsar Jahan, Ramesh Krishnamurti and Apurva Mugdal. A $\sqrt{d}$ approximation algorithm for covering points with axis parallel lines in d dimensions, in preparation.

[9] Richard Lipton. Branch and boundwhy does it work? December 19, 2012.

[10] L. Lovasz. *On Minimax Theorems of Combinatorics*. PhD thesis, 1975(in Hungarian).

[11] C. H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. 1982.

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

[13] Michael Sipser. *Introduction to The Theory of Computation*. 2006.

[14] James K. Strayer. *Linear Programming and Its Applications*. 2012.

[15] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms*. 2001.

[16] Luca Trevisan. *CombinatorialOptimization: Exact and Approximate Algorithms*. March 19 2011.

[17] Robert J. Venderbei. *Linear Programming: Foundations and Extensions*. 2001.