AN EXACT ALGORITHM FOR OPTIMAL AREAL POSITIONING PROBLEM WITH RECTANGULAR TARGETS AND REQUESTS

A Thesis

by

MANISH BANSAL

Submitted to the Office of Graduate Studies of
Texas A&M University
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Industrial Engineering

AN EXACT ALGORITHM FOR OPTIMAL AREAL POSITIONING PROBLEM WITH RECTANGULAR TARGETS AND REQUESTS

A Thesis

by

MANISH BANSAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Kiavash Kianfar
Committee Members,	Wilbert Wilhelm
	Thomas Schlumprecht
Head of Department,	Brett Peters

December 2010

Major Subject: Industrial Engineering

ABSTRACT

An Exact Algorithm for Optimal Areal Positioning Problem with Rectangular Targets and Requests (December 2010)

Manish Bansal, B.Tech., N.I.T.K. India

Chair of Advisory Committee: Dr. Kiavash Kianfar

In this thesis, we introduce a new class of problems, which we call Optimal Areal Positioning (OAP), and study a special form of these problems. OAPs have important applications in earth observation satellite management, tele-robotics, multi-camera control, and surveillance. In OAP, we would like to find the optimal position of a set of floating geometric objects (targets) on a two-dimensional plane to (partially) cover another set of fixed geometric objects (requests) in order to maximize the total reward obtained from covered parts of requests. In this thesis, we consider the special form of OAP in which targets and requests are parallel axes rectangles and targets are of equal size. A predetermined reward is associated with covering an area unit of each request. Based on the number of target rectangles, we classify rectangular OAP into two categories: Single Target Problem (STP) and Multi-Target Problem (MTP). The structure of MTP can be compared to the planar p-center which is NP-complete, if p is part of the input. In fact, we conjecture that MTP is NP-complete. The existing literature does not contain any work on MTP. The research contributions of this thesis are as follows:

- We develop new theoretical properties for the solution of STP and devised a new solution approach for it. This approach is based on a novel branch-and-bound (BB) algorithm devised over a reduced solution space. Branching is done using a clustering scheme. Our computational results show that in many cases our approach significantly outperforms the existing Plateau Vertex Traversal

and brute force algorithms, especially for problems with many requests appearing in clusters over a large region.

- We perform a theoretical study of MTP for the first time and prove several theoretical properties for its solution. We have introduced a reduced solution space using these properties. We present the first exact algorithm to solve MTP. This algorithm has a branch-and-bound framework. The reduced solution space calls for a novel branching strategy for MTP. The algorithm has a main branch-and-bound tree with a special structure along with two trees (one for each axis) to store the information required for branching in the main tree in an efficient format. Branching is done using a clustering scheme. We perform computational experiments to evaluate the performance of our algorithm. Our algorithm solves relatively large instances of MTP in a short time.

DEDICATION

This thesis is dedicated to all my teachers, particularly to my first teacher and one of the best teachers in my life, my Mother, Mrs. Raksha Bansal. She helps me to maintain a constant focus for my education and career development. It is also dedicated to my father, Mr. YashPal Bansal, who teaches me the lessons of hard work and honesty. He has inculcated in me the capability to solve wide variety of problems in real life.

This thesis is also dedicated to my younger brother, Aayush Bansal, my paternal uncles, Mr. Subhash Bansal and Prof. C.P. Bansal, and all other family members for their unstinted support. It is also dedicated to all my math's teachers in high school especially Mr. Shyam, Mr. Narender Gupta and Ms Meera, and all my friends. Finally, I would like to dedicate this thesis to my advisor, Dr. Kiavash Kianfar, for inspiring me to take challenging research. He has taught me to make my way through the toughest problems in the world, not only in mathematics but also in real life.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Kianfar, and my committee members, Dr. Wilhelm, and Dr. Schlumprecht, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues for their encouragement. Special thanks to my friend, Chandan Aggarwal, for introducing me to the webpage of Dr. Song.

Finally, thanks to my department faculty and staff, external faculty and campus employers for making my time at Texas A&M University a great experience.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1. Optimal Areal Positioning (OAP): An Introduction

We define Optimal Areal Positioning (OAP) as a class of problems in geometric optimization. In OAP, we would like to find the optimal positions of k floating geometric objects (targets) on the two-dimensional plane to (partially) cover another set of n_r fixed geometric objects (requests) in order to maximize the total reward obtained from covered parts of requests. We denote request i by r_i and target j by t_j. The coverage reward is calculated as follows: For any request r_i, $r_i \cap \left(\cup_{j=1}^{k} t_j\right)$ is the covered part of the request (by one or more targets) and $A\left(r_i \cap \left(\cup_{j=1}^{k} t_j\right)\right)$ denotes the area of this covered part. Then the reward obtained from this request will be $v_i A\left(r_i \cap \left(\cup_{j=1}^{k} t_j\right)\right)$, where v_i is the reward rate of request i, i.e. the reward per area unit of the request. The goal is to find the *optimal position* of the targets such that the total reward obtained from the coverage of the requests by the targets is maximized.

In this thesis, we address particular forms of OAP in which the targets and requests are of rectangular shape with axes parallel to x and y axes and the targets are of equal size. Based on the number of targets, this form is further classified into two categories: 1) Single Target Problem (STP), in which we consider the positioning of only one target rectangle; 2) Multi Target Problem (MTP), in which we consider positioning multiple target rectangles. The STP and MTP will be formally defined in detail in Chapters III and IV, respectively.

This thesis follows the style of the series *Lecture Notes in Computer Science* (*LNCS*).

OAP in general, and STP/MTP in particular, are motivated by several applications including camera frame selection in earth observing satellites and controlled surveillance cameras (Chapter II). Additionally, these problems have applications in what we refer to as areal facility location, where instead of aggregating the customers in centroids, the facilities and customers are considered as areas and the amount of coverage (service) to each customer is determined by the overlap of the facilities' range of coverage with that customer's area.

The only previous work directly related to STP is the work of Song et al. [29]. In fact they proposed their algorithms for an extension of this problem in the earth observing satellite context. They presented a brute force search algorithm and introduced another algorithm called plateau vertex traversal (PVT) which works better than brute force. They showed that the PVT algorithm is of complexity $O(n_r^2)$ while the brute force approach is $O(n_r^3)$. Based on our literature review, MTP has not been addressed before. Latest research in geometric optimization, particularly rectangle coverage problems and related applications is reviewed in Chapter II.

1.2. Contributions

- In this thesis we develop new theoretical properties for the solution of STP and devise a new solution approach for it (refer Chapter III). This approach is based on a novel branch-and-bound (BB) algorithm devised over a reduced solution space. The solution space is reduced based on the derived theoretical properties of the optimal solution. Branching is done using a clustering scheme.

- We present computational experiments on our algorithm for STP. The results show that in several cases our approach significantly outperforms the existing Plateau Vertex Traversal, especially for problems with many requests appearing in clusters over a large region.

- We perform a theoretical study of MTP and prove several theoretical properties for its solution (we conjecture that MTP is NP-complete and that it can be proved by reducing the planar p-center problem to MTP). We introduce a reduced solution space using these properties.

- We present the first exact algorithm to solve MTP. This algorithm has a branch-and-bound framework. The reduced solution space is essential to our algorithm and calls for a novel branching strategy for MTP. The algorithm has a main branch-and-bound tree with a special structure along with two trees (one for each axis) to store the information required for branching in the main tree in an efficient format. Therefore it is memory and performance efficient. Branching is done using a clustering scheme. Our algorithm is capable to quickly concentrate on regions with higher chance of containing the optimal solution. Based on our literature review no work has been done so far on MTP and our theoretical results and algorithm is the first attempt to solve the problem exactly and efficiently.

- We perform computational experiments to evaluate the performance of our algorithm. Our algorithm solves relatively large instances of MTP in a short time. Using our algorithm, in average problems with two targets and 50 requests are solved in about 1 second. Problems with three targets and 25 requests are solved in about 90 seconds, and problem with four targets and 10 requests are solved in about 19 minutes. For larger problems the time of our algorithm increases to more than an hour, which is still an extremely small fraction of a brute force search. The algorithm is not only fast but it is also memory efficient. Although the number of nodes shown in the computations is very large but the maximum number of node open during execution of is very small using a depth first search strategy that only creates the nodes when they are to be considered.

1.3. Organization of the Thesis

The thesis organization is as follows: In Chapter II we will review the applications that motivate our problems. We review the previous research done to solve STP and other problems in satellite imaging, such as selecting and scheduling the images to be taken by an earth observing satellite, as well as camera security surveillance. We also present a brief survey of some other related applications. We also present the related theoretical work done in geometric optimization.

In Chapter III we define STP and provide a new solution approach to solve the problem exactly. We present the theoretical properties that we use in designing our branch-and-bound algorithm. We then present the different component of our algorithm and their pseudo codes. We also present our computational experiments and discuss the performance of our algorithm versus PVT algorithm.

In Chapter IV we introduce MTP and perform a theoretical study of the problem. We prove several theoretical properties for its solution. We then present our novel branch-and-bound algorithm to solve MTP exactly along with the pseudo codes of its several components. We provide an example to explain the algorithm and present our computational experiments on our algorithm to show that it efficiently solves large instances of the problem.

In Chapter V we conclude by a summary and a discussion of several paths for future research.

CHAPTER II

APPLICATIONS, BACKGROUND AND RELATED WORK

The STP and MTP are geometric optimization problems which determine optimal placement of k rectangles on a plane to cover (partially) requests in order to maximize total reward. Although, researchers have worked on the problems involving rectangle coverage, fitting and intersection, we will see that the previous research is limited to a single study on the STP and no work has been done before to solve the MTP.

Subsection 2.1 covers the applications which are the motivation behind our problems. We review the research done to solve problems in satellite imaging, i.e. selecting and scheduling the images to be taken by an earth observing satellite, camera security surveillance, and briefly give a survey of some other related applications. In Subsection 2.2, we discuss some related theoretical work done in geometric optimization.

2.1. Motivation and Applications

The STP and MTP are motivated by applications in camera frame selection in earth observing satellites and security cameras installation for surveillance. We discuss these applications in the following subsections.

2.1.1. Earth Observing Satellite Problems

The mission of an Earth Observing Satellite is to acquire images of specified areas on the Earth surface, in response to observation requests from customers. Perhaps the closest work to the topic of this thesis is that of Song et al. [29]. They proposed algorithms for an extension of this problem in the earth observing satellite context. Several simultaneous requests for photographing a region (issued by different users) are sent to the satellite camera and a single imaging frame must be chosen for the camera.

The reward from covering one area unit of a request by the frame depends on the relative importance of the user. The position of a target frame for the camera is to be chosen such that the total reward obtained is maximized. In their version of the problem the reward is not only related to the amount of coverage but also to the ratio of the requested resolution to the actual image resolution. If the resolution factor is removed their problem reduces to the STP. Therefore their algorithms also solve the STP. They presented a brute force search algorithm and introduced another algorithm called Plateau Vertex Traversal (PVT) which works better than brute force. They showed that the PVT algorithm is of complexity $O(n_r^2)$ while the brute force approach is $O(n_r^3)$ where the number of requests is n_r.

Song and Goldberg [30] proposed an approximation algorithm for generalized version of STP i.e. when requests are not necessarily rectangular. The algorithm runs in $O(\frac{n}{\varepsilon^3})$ time where number of requests is n and approximation bound is ε. Main differences between their generalized version of problem and STP [29] are shape of input, computational speed and accuracy. In [30], input requests are not necessarily rectangular as considered in [29] and speed is preferred over accuracy.

In case of the STP, the PVT algorithm of [29] works great when the number of requests is small or moderate; however as we will see in the Chapter III, as the number of requests gets larger the branch-and-bound algorithm that we have proposed for STP works significantly faster. Xu and Song [37] have addressed p-frame problem, an extension of single frame problem [29] for p camera frames. They assumed that the p camera frames have no overlap on their coverage and a request is satisfied only if it is fully covered by a camera frame. They developed a lattice based approximation algorithm to solve the p-frame problem in $O\left(\frac{n}{\varepsilon^3}+\frac{p^2}{\varepsilon^6}\right)$ time for a given approximation bound ε. This work is merged with a paper [38] to provide a complete algorithm for request assignment and the camera parameter selection problems, and system design for autonomous surveillance [39]. In 2010, Xu et. Al. [40] proposed exact algorithms to

solve 2-frame problem in $O(n_r^2)$, $O(n^2 m)$ and $O(n^3)$ times for fixed, m discrete and continuous camera resolution levels, respectively. If the resolution factor and all the assumptions are removed, their problems [37, 39] convert to MTP which will be solved exactly in Chapter IV.

Some other problems related to earth observing satellites have been studied too. Hall et al [15] studied the satellite space mission scheduling problem for non-agile satellites (SPOT5): Given a set of jobs on a satellite (each having fixed duration, an available time window, and a weight), the goal is to schedule the jobs by selecting a feasible sequence of jobs which maximizes the sum of weights. They argued that the problem is NP-complete since it is a generalization of the problem of sequencing with release times and deadlines. Gabrel [12] proposed to formulate the scheduling problem (referred to as the shot sequence problem in [12]) using mathematical programming and graph theory. In order to obtain approximation solutions and better upper bounds, the problem can be translated into sum of simple longest paths problems as sub-problems. The formulation in [12] assumes that any shot can be taken by only one camera at most once and at a unique moment.

Vasquez and Hao [33] presented a formulation of the daily photograph scheduling problem as a generalized version of the knapsack model, followed by development of a tabu search algorithm. Later, they [34] introduced tight upper bounds. These bounds are obtained with a partition-based approach following the "divide and conquer" principle. The management of Agile Earth Observing Satellites (AEOS), which has two additional degrees of freedom i.e. a three-axis robotic camera that can be steered during each time window, has been investigated by Lemaître et al.[23]. They have presented different methods: greedy algorithm, dynamic programming algorithm, constraint programming approach and local search method.

2.1.2. Security Cameras

Security cameras mounted on different structures (buildings, towers, etc.) are important tools being used more and more everyday to monitor public and commercial facilities against various threats. Millions of dollars are being spent by local governments on installing these security systems. Pre-installed cameras are also used in other contexts such as environmental research, traffic control, border protection, etc. Availability of cameras is clearly subject to budget constraints. Therefore in practice adjusting the view-frames of the limited number of available cameras (i.e. where they are looking) to optimally monitor a large region becomes an important (and complex) problem. This problem arises in both manual and automatic control of cameras.

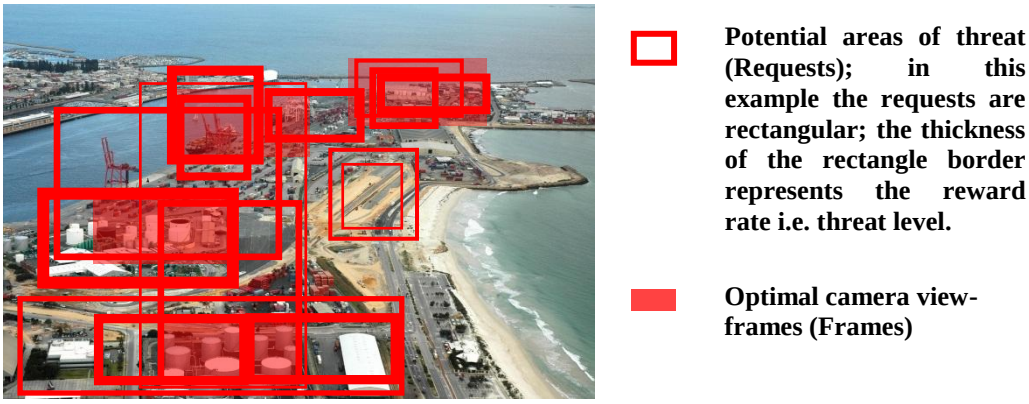


Figure 2.1. Finding the view-frames for 4 security cameras in surveillance of a port using OAP (the cameras will pan-tilt-zoom onto the designated optimal frames)

OAP can be directly applied to address problems of this nature. Here we describe an example in the context of port security (Fig. 2.1). Consider a certain number of (pan-tilt-zoom) security cameras installed on one or more towers to monitor a part of a large port. The view-frames of these cameras are controlled by an automatic system. We would like the system to automatically adjust the view-frames of cameras to optimally monitor the

port in order to minimize the possibility of missing a threat. This problem can be formulated as an OAP. Based on the activities going on in the port for a given time window, human agents or preprogrammed automatic systems identify (as input to the system) several areas of potential threat on a panoramic picture of the port. These areas represent the "requests" in the OAP. They are the bordered rectangles in Fig. 2.1. The threat level of different areas may be different depending on their sensitivity, volume of activity, and vulnerability. A reward rate is associated to each request proportionate to the level of threat in that area in that time window. The sought-after camera view-frames are the "targets" in the OAP. By solving the OAP, the automatic control system finds the optimal positions of view-frames (color-filled rectangles in Fig. 2.1) such that the total level of threat that they cover is maximized. Then the cameras pan-tilt and zoom on those view-frames.

2.1.3. Other Related Applications

There are other problems related to covering. Identifying the minimum number of discs with fixed radius to cover a given set of points in the plane is an example. This problem has been addressed as NP-hard problem in a number of articles. Hochbaum and Mass [14] presented polynomial approximation algorithms for different versions of geometric covering problems, including covering by discs. Agnetis et al. [2] addressed the disc covering problem on a line - the problem of covering (or full surveillance of) a single line segment with radar sensors having a circular field of view at minimum cost. This has been referred as the robust k-center problem and analyzed in [2]. Agnetis et al. mentioned that for identical radius sensors, a simple polynomial search solves for optimal radius and number of sensors. But the problem becomes hard when the sensors are modeled with variable diameter discs.

The deployment of wireless transmission networks is related to the geometric Disc Covering Problem. Surveys on covering problems dealing with this particular application can be found in [7, 20, 32]. Huang and Tseng [20] surveyed the solution to

the following covering problems in wireless sensor networks: 1) The *Art Gallery Problem* introduced by Chvátal [7], where one has to find the minimum number of watchmen (or cameras) needed to observe every wall of an art gallery room. 2) Energy-conserving protocols and coverage-preserving sensor scheduling scheme which determine when a sensor node can be turned off and when it should be rescheduled to become active again. 3) Surveillance issues of achieving certain sensing coverage and communication connectivity requirements and evaluating the quality of service provided by a particular sensor network. 4) The Circle Covering Problem which is to arrange identical circles on a plane that can fully cover the plane. Thai et al. [32] have presented an overview of coverage problems in wireless sensor networks. We reviewed several papers dealing with this particular application as the disc coverage problem is defined from several points of view due to a wide-range of applications. Among all the problems, only the k-coverage problem in [32] sounds closer to our MTP. But the k-coverage problem deals with finding a set of sensors such that every point in an area is covered by at least k- distinct sensor nodes, which is completely different from the definition of the MTP.

2.2. Related Theoretical Work in Geometric Optimization

The STP and MTP are problems belonging to the class of geometric optimization problems. Lu. et al. [24] have addressed several problems belonging to the class of rectangle intersection in computational geometry. They solved following problems for a set of n rectangles: a) calculating the area of the region that is covered by at least one rectangle or by two or more rectangles, b) finding the maximum number of rectangles that overlap and c) calculating the distance between the closest pair of non overlapping rectangles. The algorithms presented in their paper employ a divide-and-conquer technique.

In 1998, Agarwal and Sharir [1] reviewed the progress in the design of algorithms for various geometric optimization problems in a survey. They reviewed several

techniques used to tackle the problems in geometric optimization, including facility location, proximity problems, statistical estimators and metrology, placement and intersection of polygons and polyhedral, and ray shooting and other query type problems. The techniques addressed in [1] are parametric searching, geometric alternatives to parametric searching, prune-and-search techniques. From computational point of view, Latin and Lbbecke [19] addressed the problem of covering a polygon with a minimum number of rectangles. Another rectangle placement problem is considered by Amos and Oran [21] whose goal is to find a placement of maximum number of rectangles while scheduling a sequence of rectangles on a matrix. A matrix is a rectangular area on a two-dimensional plane. They presented an $O(n \log^2 n)$ time approximation algorithm. Saha and Das [27] considered the coverage of a set of n points on a plane by two parallel rectangles placed in arbitrary orientation, such that the area of the larger rectangle is minimized. They solved the problem in $O(n^3)$ time using an $O(n^2)$ space. The problem of minimizing the total area of two rectangles placed to cover a given set of points on a plane can also be solved by the approach in [27]. Ahn and Bae [3] extended the two-rectangle covering problem by considering: 1) the rectangles are free to rotate but must remain parallel to each other, and 2) one rectangle is axis-parallel but the other rectangle is allowed to have an arbitrary orientation. They presented $O(n^2 \log n)$ time algorithms for solving both problems, which is an improvement to the algorithm in [27].

The MTP is an extension of STP when multiple targets are to be positioned. The structure of this problem can be compared to the planar p-center or p-median problems which are NP-complete, if p is part of the input [25]. In fact, we conjecture that MTP is NP-complete by reducing p-center problem to MTP. The p-center and p-median problems deal with points instead of area though: in p-center given a set of demand points, the goal is to locate p service points on the plane to minimize the maximum distance of a demand point to its nearest service point. In p-median the goal is to minimize the summation of such distances. The 2-center problem is a special case of the

general p-center problem. This problem has been studied in several papers [6, 11, 16, 28], and the currently best algorithm for its solution runs in polynomial time [11, 16].

In the plane, p-center problem was investigated by Drezner [8] and Vijay [35] for Euclidean distances and by Dzerner [9] for rectilinear distance. The Euclidean p-center problem is equivalent to the following two related problems: 1) Covering every point in the area by p circles with the smallest possible radius. 2) Locating p objects such that the total weight of points within a fixed distance of some object is maximized. Likewise, the rectilinear p-center problem is to cover every point in the area by p squares of minimum area. A text by Handler and Mirchandani [17] addresses networks location problems corresponding to p-center extensively, and excellent reviews of the p-center problem on trees and graphs can be found in Handler [17] and Tansel et al. [31].

Another closely related problem is the p-dispersion problem, which is to locate p facilities in an area or a graph such that the minimal distance between two facilities is maximized. The p-dispersion problem in a square is equivalent to packing p circles with maximal radius in a square as discussed in Drezner and Erkut [10]. Research in this field has followed two directions. The first deals with finding packing with proven optimality and the second aims toward finding algorithms with better complexity.

CHAPTER III

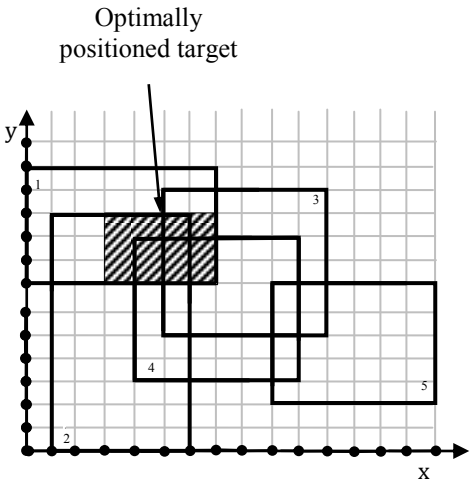
SINGLE TARGET PROBLEM

3.1. Introduction

In this chapter, we consider positioning a *single* rectangular target on two-dimensional plane to partially cover a set of existing rectangular areas (requests) to maximize total coverage reward. More specifically this problem can be described as follows (see Fig. 3.1): n_r rectangular areas, called request rectangles or simply *requests*, are designated on a two-dimensional plane. Their positions and sizes are known, i.e. for request i, denoted by r_i, we know the values $x_i, y_i, w_i,$ and l_i, where x_i and y_i are the coordinates of the *lower left* corner of the request and w_i is its width (length along x axis) and l_i is its length (length along y axis). All requests have axes parallel to x and y axes. We would like to find the optimal position of a floating target rectangle (simply called *target* and denoted by t), with the known dimensions w_t and l_t *and* axes parallel to x and y axes, such that the *coverage reward* obtained from this positioning is maximized. The coverage reward is calculated as follows: Let $r_i \cap t$ denote the part of request i that is covered by target and $A(r_i \cap t)$ denote the area of this covered part. Then there will be a reward equal to $v_i \times A(r_i \cap t)$, where v_i is the reward rate of request i, i.e. the reward per area unit of the request. We would like to find the *optimal position* of the target (by position we mean x_t and y_t, the coordinates of its lower left corner) such that the total reward obtained from the coverage of requests by the target is maximized. In other words we want to solve the following problem:

$$\max_{x_t, y_t} \quad \sum_{i=1}^{n_r} v_i A(r_i \cap t).$$

Fig. 3.1 shows the picture and data along with the optimal position of the target for a solved problem with five requests. Based on the target position we see that the reward from request 1, 2, 3, 4 and 5 are $4 \times 12, 6 \times 9, 7 \times 6, 3 \times 6,$ and $10 \times 0,$ respectively so the total reward is 162. It is important to note that in this thesis, other than the



request r	lower left corner (x_r, y_r)	width w_r	length l_r	reward rate v_r
1	(0,7)	7	5	4
2	(1,0)	5	10	6
3	(5,5)	6	6	7
4	(4,3)	6	6	3
5	(9,2)	6	5	10

details of target		optimal	optimal
width (w_t)	length (l_t)	position (x_t, y_t)	(max) reward R
4	3	(3,7)	162

Figure 3.1: An illustrative example of the coverage problem with a single target and its optimal solution

examples, there is no grid and all the problems are exactly in continuous space i.e. there is no discretization.

As discussed in Chapter II, STP is motivated by several applications including camera frame selection in earth observing satellites or controlled surveillance cameras. See Chapter II for more details. In addition to this application we believe this problem can have application in what we refer to as areal facility location where instead of aggregating the customers in centroids, the facility and customers are considered as areas and the amount of coverage (service) to each customer is determined by the overlap of the facility area with that customer's area.

In this chapter we provide a new solution approach for STP. This approach is based on a novel branch-and-bound (BB) algorithm devised over a reduced solution space. The solution space is reduced based on some theoretical properties of the optimal solution. Similar theoretical properties are used by Song et al. [29]. As discussed in Chapter II, they presented a brute force search algorithm and introduced another algorithm called

plateau vertex traversal (PVT) which works better than brute force. They showed that the PVT algorithm is of complexity $O(n_r^2)$ while the brute force approach is $O(n_r^3)$. The PVT algorithm works great when the number of requests is small or moderate; however as the number of requests gets larger our extensive computational experiments show that our algorithm works significantly faster. This is especially true when the requests are scattered over a large area as our BB algorithm tends to focus quickly to the regions where there is a concentration of requests and higher rewards. For the same reason our algorithm tends to work significantly faster when the reward rates are more heterogeneous among the requests.

In Subsection 3.2, we discuss the general framework of a BB algorithm. In Subsection 3.3, we present our theoretical structure of our algorithm and in Subsection 3.4 we present our computational experiments and discuss the performance of our algorithm versus PVT algorithm.

3.2. Structure of Branch-and-Bound Algorithms

Branch-and-bound (BB) is the most widely used tool for solving large scale *NP*-hard combinatorial optimization problems. In 1960, Land and Doig [22] were the first to propose this method for integer programming.

The schematic behind the algorithm for a maximization problem is as follows: The algorithm starts at the root node. The BB tree is a decision tree. Each node in the tree corresponds to a subset of the solution space. At each node, two main actions are performed: 1) *bounding*, this procedure calculates the upper bound for the best solution value obtainable in the solution space of each node from the tree. 2) *decision making,* based on the upper bound at a node and best known feasible solution value (i.e. best lower bound of the problem), the node is either *pruned* or *branched*.

Pruning Step: A node can be pruned for two reasons: 1) if the upper bound value on that node is smaller than the best feasible solution value found so far. In this case there is

no point in searching the node for optimal solution anymore (this is the main idea behind BB). 2) if a solution is found, the lower bound will be updated if this solution has a larger objective value.

Branching Step: If a node cannot be pruned, the solution space of the node is subdivided into two or more subspaces (by generating child nodes). This action is known as branching. There are different problem dependent strategies for choosing the branching scheme in a node and also for choosing the next node in the tree.

The problem is solved when all nodes are pruned and the best lower bound will be the optimal value. BB often leads to exponential time complexities in the worst case but if applied carefully, it can lead to algorithms that run reasonably fast on average. The efficiency of the method depends strongly on the branching (node-splitting procedure) and on the upper and lower bound estimators. In order to solve maximization problem using BB, interchange the lower bound by upper bound in the scheme mentioned above. More details and references can be found in [26] and [36].

3.3. Novel Branch-and-Bound Algorithm for STP

In this subsection we present our algorithm for solving the STP problem. We mentioned in Subsection 3.1 that we determine the positions of requests and the target by the coordinates of their lower left corner. We will use a fundamental observation (Lemma 3.1) in our BB algorithm. This observation helps us to reduce the continuous two-dimensional solution space to a set of discrete points. To identify these discrete points, first we define what we will refer to as *critical x and y values.* Given a request i, four critical x values and four critical y values corresponding to this request are defined as follows (see Fig. 3.2):

$$x_i^1 = x_i - w_t; \quad x_i^2 = x_i; \quad x_i^3 = x_i + w_i - w_t; \; x_i^4 = x_i + w_i$$

$$y_i^1 = y_i - l_t; \quad y_i^2 = y_i; \quad y_i^3 = y_i + l_i - l_t; \quad y_i^4 = y_i + l_i$$

These critical values are shown for two requests in Fig. 3.2. Since the critical values are generated from requests so we call them RCVs. The RCVs are numbered from left to right from 1 to 4. We categorized RCVs into two types: RCVs 2 and 3 are type 1 (denoted by RCV1) and RCVs 1 and 4 are type 2 (denoted by RCV2). According to Fig

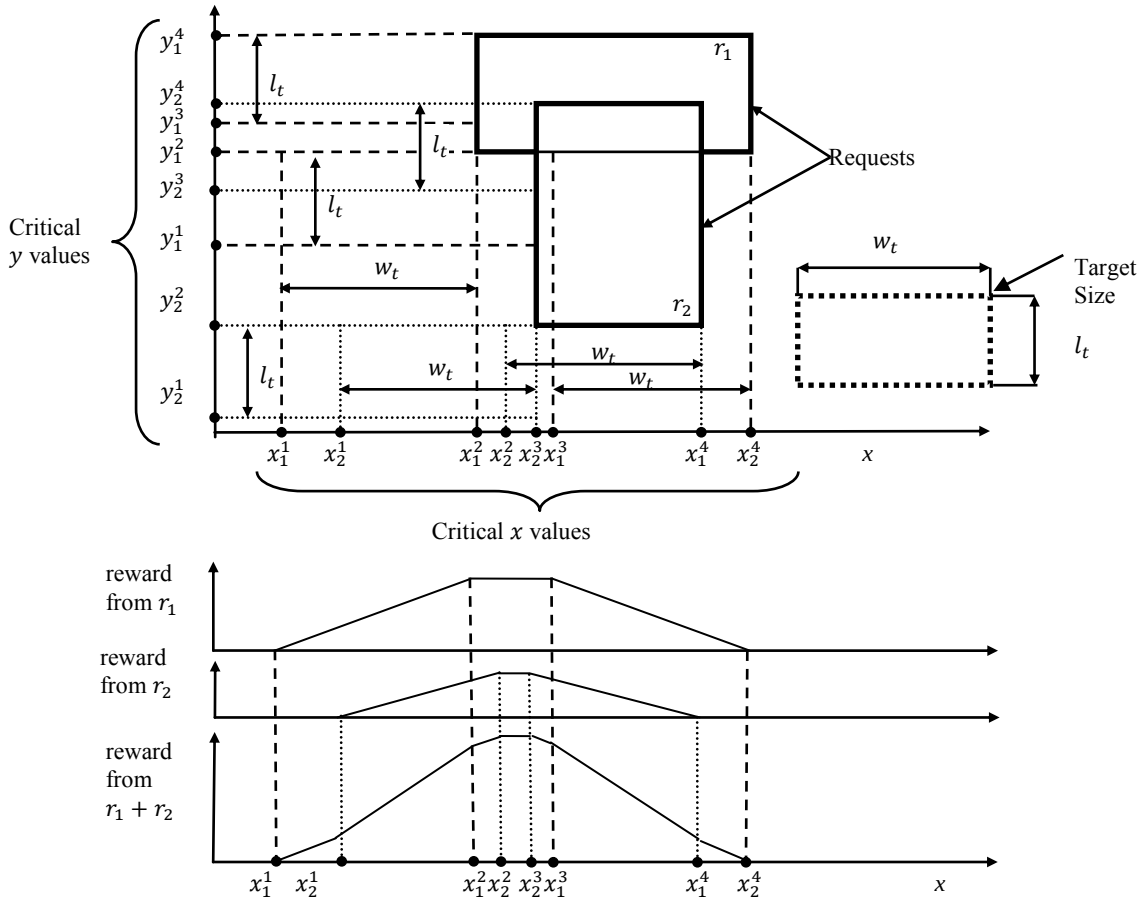


Figure 3.2. x and y RCVs for $r_1$and r_2 and reward graphs at $y_t = y_1^2$

3.2, $x_i^2, x_i^3, y_i^2, y_i^3$ are RCV1s and $x_i^1, x_i^4, y_i^1, y_i^4$ are RCV2s. The graph at the bottom of the figure shows the reward function when the target moves parallel to the x axis with y_t at y_1^2 for this example. It is easy to see in Fig. 3.2 that the reward function for a fixed value of y_t is piecewise linear and its breakpoints occur at the x RCVs. A similar statement is true for a fixed x_t and critical y values. This is because the area covered by the target with one of its coordinates fixed is simply a linear function of the other coordinate. Now the fundamental result that we use to reduce the solution space to discrete points is as follows. A result similar to this is used in [29] in design of brute force and PVT algorithm.

Lemma 3.1. *There is at least one optimal position (x_t, y_t) for the target such that x_t is an x RCV1 and y_t is a y RCV1.*

Proof. Assume the position $(x_t, y_t) = (x_t^*, y_t^*)$ is an optimal position for the target and x_t^* is not an x RCV1. Notice that x_t^* cannot be an x RCV2 because if it is, then by moving the target parallel to x axis, either left or right, the reward increases, which is contrary to optimality of x_t^*. Now, by moving the target to the left or right with y_t fixed (i.e. parallel to the x axis) until x_t is equal to an x RCV1, the reward will not change because if the reward increases it contradicts optimality of x_t^*; if the reward decreases, since the reward function at a fixed y is piecewise linear with breakpoints at x RCVs, by moving in the opposite direction, the reward increases, which is again contrary to optimality of x_t^*. Therefore the reward will remain the same and we will hit an x RCV. Notice that this RCV cannot be a RCV2 because, as explained above, if it is a RCV2, then by moving the target parallel to x axis, either left or right, the reward increases, which is contrary to optimality of x_t^*. Therefore we have found a new optimal solution in which x_t is an x RCV1.The same argument can be applied to y_t. Therefore there will be another optimal solution with both x_t and y_t being RCV1s. □

As a result of Lemma 3.1, we only need to do our search over the points that are at the intersection of x and y RCVs (we call them *critical points* (CPs)). The Brute Force

BBAlgorithm

[Initialization]

1. $Input\ set\ of\ requests\ R_0 := \{r_i[x_i, y_i, w_i, l_i, v_i],\ i = 1, \dots, n_r\};$
2. $Initialize\ BBtree, P_0 := Root\ BBnode;$
3. $X_{P_0} := C_x; Y_{P_0} = C_y; LB := \max_{i=1,\dots,n_r} S_{r_i}$;

[End of Initialization]

4. $P := P_0;$
5. $While\ P \neq Null$
6. $If\ UB(P) \leq LB\ then$

[Upper Bound at node P is less than the best solution value known so far]

7. $NextNode(P);$
8. $Else\ If\ |X_P| = 1\ and\ |Y_P| = 1\ then$ [The target has fixed x, y coordinates]
9. $LB := UB(P);\ Solution := (X_P, Y_P);$
10. $NextNode(P);$
11. $Else$
12. $Branch(P);$
13. $P := Child(P, 1);$
14. $End\ if$
15. $End\ While$
16. $Optimal\ Value = LB;$

Figure 3.3. Main body of the BB algorithm

Algorithm does this by calculating the reward function for every CP. At each CP the coverage of each request is calculated and the associated rewards are added and the critical point with the highest reward is found. The number of CPs is $O(n_r^2)$ and reward calculation for each is $O(n_r)$. So the complexity is $O(n_r^3)$, which is not efficient at all when n_r is a large number. The PVT algorithm [29] does a better job of calculating the reward. Instead of calculating the reward at each CP independently, it scans the CPs in movements parallel to one of the axes and finds the reward at the next CP by calculating the slope of the piecewise linear functions. This reduces the complexity to $O(n_r^2)$.

One drawback of the PVT algorithm is it still has to scan through all CPs. If there is a higher concentration of requests or large reward rates at some region of the plane, the chances of the optimal solution being in that region is higher. Our BB algorithm is designed to benefit from this feature. It quickly concentrates on such regions and finds a

good solution and then by calculating upper bounds on the reward if the target is in other regions tries to avoid explicit enumeration of CPs in those regions.

Our BB algorithm implicitly searches the space of CPs. Like any BB algorithm (see [4]), our algorithm is performed over a tree, which we call BBtree, and uses bound to avoid explicitly enumerating all CPs. We denote the set of all x and y RCV1s by C_x and C_y, respectively. The nodes of this tree are called BBnodes. Each BBnode P in BBtree is associated with a subset X_P of x RCV1s and a subset Y_P of y RCV1s. Therefore each node in fact corresponds to a rectangular mesh consisting of a subset of CPs. These subsets are created by clustering the sets $C_x$and C_y on two clustering trees. When required, the set of RCV1s in each cluster is further clustered (partitioned) into subsets. The subsets X_P and Y_P at each BBnode P correspond to a cluster of x RCV1s and a cluster of y RCV1s respectively. The pseudo code for the main body of our BB algorithm is shown in Fig. 3.3. LB denotes the lower bound on maximum reward and S_{r_i} is the maximum reward possible from covering request i by the target if target can be placed anywhere, i.e. $S_{r_i} = v_i \max_{(x_t,y_t)} A(r_i \cap t)$.

The pseudo codes of the functions used within the BB algorithm are presented in Fig. 3.4. Function $UB(P)$ calculates the reward upper bound at each node (an upper bound on the reward when the target position is limited to the CPs corresponding to that node). $R(P)$ denotes a set of trimmed requests for node P. These are obtained by original requests that are trimmed to the region that is reachable by the target when its position is limited to the region associated with the CPs of node P. Based on definition of S_r, clearly summation of S_r for all $r \in R(P)$ is an upper bound for reward at node P and that is the way we calculate $UB(P)$ in our algorithm $NextNode(P)$ finds the next node after pruning a node based on a depth first search strategy. Other search strategies can also be implemented in this function.

$\boldsymbol{UB}(\boldsymbol{P})$:
1. $For\ r \in R(Parent(P))$
2. $\quad Trim\ request\ r\ to\ a\ new\ request\ r'such\ that\ every\ point\ of\ r'is\ coverable$
3. $\quad by\ the\ target\ whose\ position\ is\ limited\ to\ the\ region\ of\ node\ P;$
4. $\quad R(P) := \{R(P) \backslash r\} \cup r';$
5. $End\ for$
6. $For\ r \in R(P)$
7. $\quad UB = UB + S_r;$
8. $End\ For$

$\boldsymbol{NextNode}(\boldsymbol{P})$:
9. $If\ RightSibling(P) \neq Null\ then$
10. $\quad NextNode := RightSibling(P)$; [Move Right on BBTree]
11. $Else\ if\ Parent(P) \neq Null\ then$
12. $\quad P := Parent(P); NextNode(P);$ [Move Up on BBTree]
13. $Else$ [Current node is root node]
14. $\quad Nextnode := Null;$
15. $End\ if$

$\boldsymbol{Branch}(\boldsymbol{P})$:
16. $Using\ a\ selection\ rule\ select\ either\ X_P\ or\ Y_P$
17. $(say\ X_P\ is\ selected; the\ case\ for\ Y_P\ is\ similar)$
18. $If\ GetCluster(X_P, 1) = Null\ then\ Cluster(X_P); End\ if$
[Create clusters for x clustering tree]
19. $For\ l = 1, \dots, NumofClusters(X_P)$
20. $\quad Create\ BBnode\ P_l;\ X_{P_l} := GetCluster(X_P, l); Y_{P_l} := Y_P;$
21. $\quad Child(P, l) := P_l;$
22. $End\ For$

$\boldsymbol{Cluster}(\boldsymbol{Z})$:
$(elements\ of\ Z\ are\ denoted\ by\ z_i, i = 1, \dots, |Z|$
$and\ are\ sorted\ in\ increasing\ order)$
23. $M = max_{i=1,\dots,|Z|}(z_{i+1} - z_i);\ j = 1;$
24. $For\ i = 1, \dots, |Z| - 1$
25. $\quad If\ z_{i+1} - z_i > \beta M\ then$ [Break at the point when distance between RCV1s $> \beta M$]
26. $\quad\quad Create\ Cluster\ C := \{z_j, \dots, z_i\};$
27. $\quad\quad List := List \cup \{C\};\ j = i + 1;$
28. $\quad End\ If$
29. $End\ For$
30. $Create\ Cluster\ C := \{z_j, \dots, z_{|Z|}\};$
31. $List := List \cup \{C\};\ Sort\ List\ based\ on\ PRIORITY\ of\ clusters;$
32. $For\ l = 1, \dots, |List|$
33. $\quad SetCluster(Z, l) := List[l];$
34. $End\ for$

Figure 3.4. Functions used in BB algorithm

In Fig. 3.3 observe that when in a node $P, UB(P)$, is not greater than LB the node is pruned and the next node is considered. Also when the target location is fixed, a better solution is found the LB is updated. Otherwise, the node is branched meaning that the CPs of that node are decomposed into smaller subsets.

The branching is done using the $Branch(P)$ function. The branches are determined by clustering the set of x or y RCV1s. A selection rule to choose between X_P or Y_P must be used. One possibility is branching over X_P until it has only one element and then branch over Y_P. If we consider the case where branching is done over X_P, the set of x RCV1s for each of its branches correspond to one of its clusters. If branching is not stopped because of pruning by bound we will reach a leaf BBnode. A leaf BBnode has one element in the set of CPs along each axis. In other words, at the leaf BBnode the target gets an exact position as the CPs give the left bottom corner's coordinates of the target.

The clustering of X_P is performed by calling the $Cluster(X_P)$ function. The clusters of each set are saved by the $SetCluster$ function because they may be used in branching of future nodes again and we do not want to redo the clustering in that case. We pick the best clustering scheme out of following five clustering schemes for branching. As seen in Fig. 3.4, clustering, according to Scheme 1, is done as follows: the RCV1s in the set are sorted. Whenever the distance between two RCV1s is greater than a predetermined percentage (β) of the maximum distance between two consecutive RCV1s (M), we break the set at that point and create a new cluster. The efficiency of the BB method depends critically on the effectiveness of the branching. An appropriate value of (β) is selected because the smaller values of β would reduce our algorithm to an exhaustive enumeration of the domain, on the other hand branching won't occur for the bigger values of β. We also assign $PRIORITY$ to the clusters and consider branches in the order of priority. The priority value we use is the ratio of summation of the reward rates of the requests whose RCV1s are in the cluster to the number of RCV1s in the cluster. All other schemes perform clustering similar to Scheme 1 till the value of M in a cluster is

greater than the dimension of the target along the corresponding axis. Afterwards, Scheme 2 and 3 create two and three new clusters, respectively with equal number of RCV1s. In Scheme 4, we break the set whenever distance between two RCV1s is equal to M. Scheme 5 creates three new clusters by breaking a cluster at points where distance between consecutive RCV1s is either equal maximum or second maximum in the cluster. When a cluster contains two RCV1s then the scheme 3 and 5 work like scheme 2 and 4 respectively. We found based on our experiments that the Scheme 1 with $\beta = 50\%$ results in the smallest run times in average. The idea is of this branching scheme is to quickly focus on the regions of the plane that are populated by the requests and thus have higher probability of containing the optimal location.

The algorithm terminates when there are no more nodes to consider. Calculation of lower and upper bounds typically results in eliminating many CPs without calculating reward for them explicitly and that is the main reason that our algorithm works faster in many large problems.

3.4. Computational Results

We generated random instances with different number of requests and compared the performance of our algorithm with PVT algorithm. The brute force algorithm is worse than both for obvious reasons. The running time of the brute force algorithm easily exceeds 5 minutes for 500 requests and several hours for 4000 requests in all cases so we eliminate it from further consideration. We generated three categories of problems in terms of relative size of target and requests. They are shown in Table 3.1. In categories A, the target is smaller or equal to average of request sizes. In category B, the target size is considerably larger than the average of request sizes. In categories C, the target size can be smaller, equal or larger than average of request sizes. The requests are randomly distributed over a square region of determined size. The random requests are generated in two steps. First, we generate three points in the region to represent locations of interest, which we call as center points. For each center point, we use a radius of interest. Then, we generate requested viewing zones. To generate a requested viewing

zone, we create six random numbers. One of them is used to determine which center point the request will be associated with. Two of them are used to generate the location of the lower left corner of the request, which is located within the corresponding radius of the associated center point. The remaining three random numbers are used to generate width, length and reward of requests.

Table 3.1. Problem categories

Category	w_t	l_t	w_i	l_i
A	8	6	Uniform[1,15]	Uniform[1,11]
B	8	6	Uniform[1,7]	Uniform[1,5]
C	4	3	Uniform[1,15]	Uniform[1,11]

In Experiment 1, the results of which are shown in Table 3.2, the total region is large (5K by 5K). The reward rates of requests are from Uniform [1,150] distribution. We observe that in many instances specially the ones with large number of requests there is a significant improvement relative to PVT. This is between 50 to 80 percent in most cases (the instances in which BB works better are shaded). Note that our computations are averaged over three runs. The time improvement is 100×(PVTtime–BBtime) / PVTtime. The large total region causes random separate concentrations of requests in some areas and our BB algorithm pays off by focusing on concentrated areas and finding good lower bounds to avoid considering other areas. If the requests are distributed over a smaller area then the PVT algorithm tends to work better as BB will not be able to quickly find a LB that is considerably better than many node upper bounds. This is what we observe in Experiment 2 (Table 3.3).

Our algorithm is capable to quickly concentrate on regions with higher chance of containing the optimal solution. This is supported by the time taken by our algorithm to

reach solution that is proved to be the optimal at the end, as shown in Tables 3.2, 3.3 and Figs. 3.5 and 3.6. In Figs. 3.5 and 3.6, T denotes the total time taken by BB algorithm to solve an instance and T_1 is the time taken by LB to reach the optimal value. It can be clearly observed that T_1 is much lesser than T, in fact in many cases T_1 is 3 to 30 percent of T. Even in Experiment 2, T_1 is lesser than PVTtime for most of the cases.

Table 3.2. Experiment 1: Instances with requests distributed over a large region (5K by 5K)

Category	No. of Requests	No. of Nodes	BB Algorithm		PVT Algorithm	Time Improvement
			Time when LB reaches Optimal value T_1	Run Time T	Run Time	
A	10	7	0	0.00	0.00	-
	100	81	0	0.03	0.04	25%
	500	329	0.06	0.49	1.00	51%
	1000	1071	0.86	1.82	4.00	55%
	2000	2015	2.23	10.34	16.00	35%
	4000	4948	0.14	38.83	64.00	39%
B	10	8	0	0.00	0.00	-
	100	104	0	0.02	0.04	50%
	500	302	0	0.22	1.00	78%
	1000	622	0	0.84	4.00	79%
	2000	1762	0.41	5.00	16.00	69%
	4000	3535	0.9	19.72	64.00	69%
C	10	15	0	0.00	0.00	-
	100	107	0	0.04	0.04	-
	500	735	0.04	0.67	1.00	33%
	1000	1948	0.95	1.72	4.00	57%
	2000	6313	3.49	11.65	16.00	27%
	4000	14165	1.07	42.71	64.00	33%

Table 3.3. Experiment 2: Instances with requests distributed over a region (1K by 1K)

Category	No. of Requests	No. of Nodes	BB Algorithm		PVT Algorithm
			Time when LB reaches Optimal Value T_1	Run Time T	Run Time
A	10	7	0	0.00	0.00
	100	81	0.01	0.08	0.04
	500	329	1.02	2.20	1.00
	1000	1071	2.87	13.03	4.00
	2000	2015	5.26	41.97	16.00
	4000	4948	120.63	188.54	64.00
B	10	8	0	0.00	0.00
	100	104	0	0.05	0.04
	500	302	0.02	0.98	1.00
	1000	622	0.78	5.75	4.00
	2000	1762	13.05	39.36	16.00
	4000	3535	52.35	204.33	64.00
C	10	15	0	0.00	0.00
	100	107	0.02	0.08	0.04
	500	735	0.55	1.54	1.00
	1000	1948	5.42	11.32	4.00
	2000	6313	9.22	50.58	16.00
	4000	14165	88.29	199.37	64.00

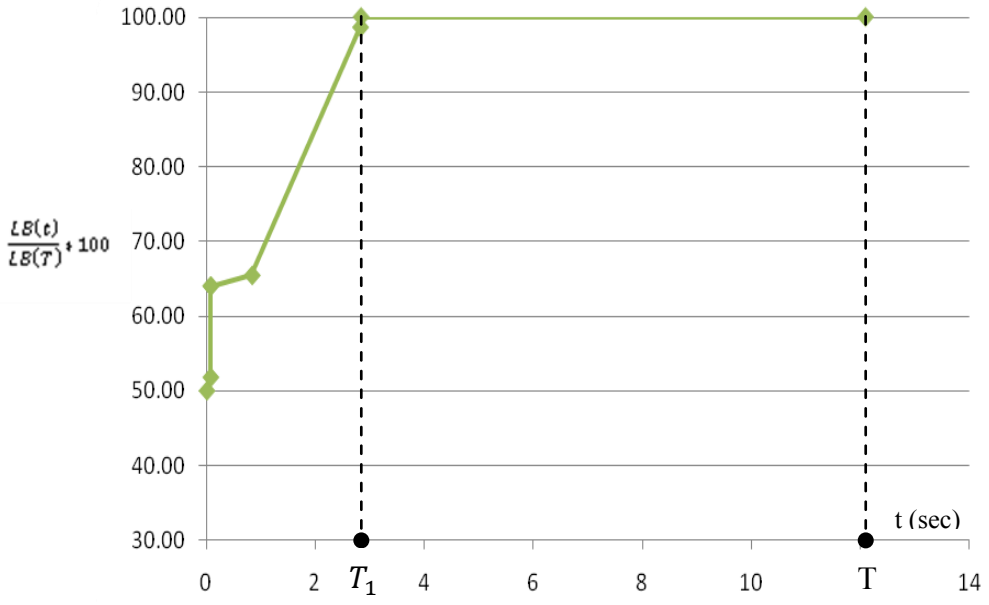


Fig. 3.5 Graph of Lower bound improvement versus time for an instance

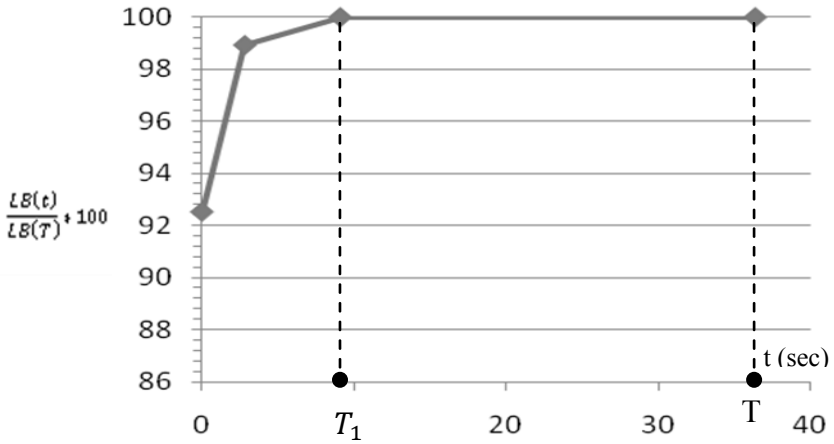


Fig. 3.6 Graph of Lower bound improvement versus time for another instance

CHAPTER IV

MULTI-TARGET PROBLEM

4.1. Introduction

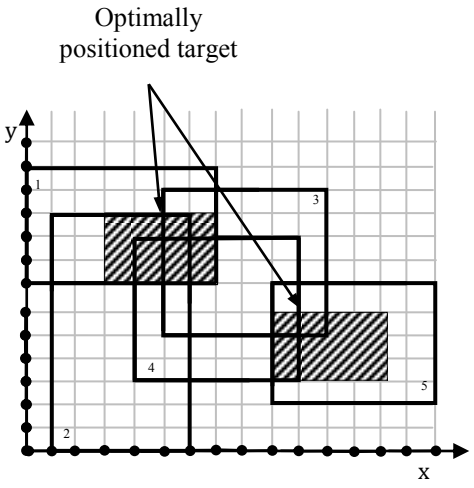
In this chapter, we address the multi-target (MTP) problem defined in Chapter I. MTP was defined as the problem of positioning k target rectangles on the two-dimensional plane to partially cover a set of existing rectangular areas (requests) to maximize total coverage reward. MTP has not been addressed in the literature before and here we perform the first theoretical study on this problem and present the first exact algorithm for MTP. More specifically this problem can be described as follows (see Fig. 4.1): n_r rectangular areas, called request rectangles or simply *requests*, are designated on a two-dimensional plane. All requests have axes parallel to x and y axes. Their positions and sizes are known, i.e. for request i, denoted by r_i, we know the values $x_i, y_i, w_i,$ and l_i, where x_i and y_i are the coordinates of the *lower left* corner of the request and w_i is its width (length along x axis) and l_i is its length (length along y axis). We would like to find the optimal position of k floating target rectangles, denoted by $t_j, j \in \{1, \dots, k\}$, (simply called *target)* such that the *total coverage reward* obtained from this positioning is maximized. The targets are also axes-parallel. The width and length of all targets are equal and denoted by w_t and l_t. The coverage reward is calculated as follows: Let $r_i \cap t_j$ denote the part of request i that is covered by target j and $A(r_i \cap t_j)$ denote the area of this covered part. Then there will be a reward equal to $v_i \times A(r_i \cap t_j)$, where v_i is the reward rate of request i, i.e. the reward per area unit of the request that is covered. Hence, we would like to find the *optimal position* of the targets (we take x_{t_j} and y_{t_j}, the coordinates of the lower left corner target j as its position) such that the total reward obtained from the coverage of requests by the targets is maximized. In other words we want to solve the following problem:

$$\max_{\substack{x_{t_j}, y_{t_j} \\ j=1,\ldots,k}} \quad \sum_{i=1}^{n_r} v_i \times A(r_i \cap (t_1 \cup t_2 \cup \ldots \cup t_k)) \, .$$

We note that in MTP, the coverage reward obtained from the covered part of a request does not depend on which and how many targets cover that part. Figs. 4.1, 4.2 and 4.3 show the pictures and data along with the optimal positions of the 2, 3 and 4 targets respectively, for the problem with five requests (We have obtained these solution using the algorithm we will describe in Subsection 4.4). Based on the targets' positions, the reward from requests 1, 2, 3, 4 and 5 are as displayed in Table 4.1.

Table 4.1. Reward contribution from each request in Figs. 4.1, 4.2 and 4.3

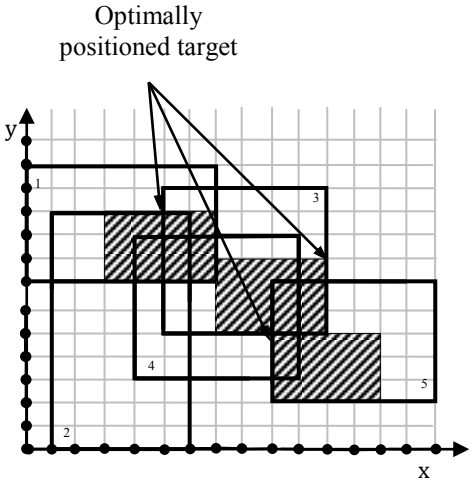
No. of Targets	Rewards from requests	Total Reward
2	4x12, 6x9 , 7x6 , 3x6 , 10x9	319
3	4x12 ,6x9 ,7x18 ,3x17 ,10x12	439
4	4x20 , 6x17 , 7x28 , 3x21 , 10x12	561



request r	lower left corner (x_r, y_r)	width w_r	length l_r	reward rate v_r
1	(0,7)	7	5	4
2	(1,0)	5	10	6
3	(5,5)	6	6	7
4	(4,3)	6	6	3
5	(9,2)	6	5	10

details of target		optimal position (x_t, y_t)	optimal (max) reward R
width (w_t)	length (l_t)		
4	3	(3,7) (9,4)	319

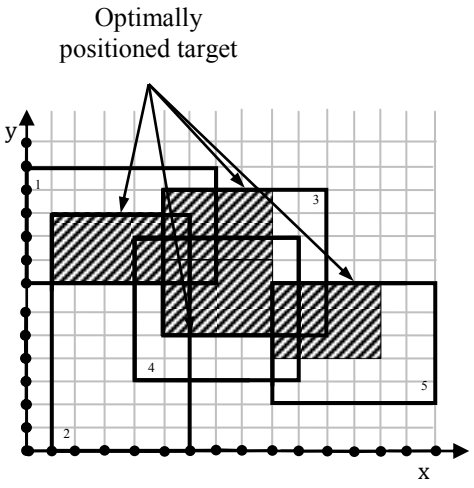
Figure 4.1. An illustrative example of the coverage problem with two targets and its optimal solution



request r	lower left corner (x_r, y_r)	width w_r	length l_r	reward rate v_r
1	(0,7)	7	5	4
2	(1,0)	5	10	6
3	(5,5)	6	6	7
4	(4,3)	6	6	3
5	(9,2)	6	5	10

details of target		optimal position (x_t, y_t)	optimal (max) reward R
width (w_t)	length (l_t)		
4	3	(3,7) (7,5) (9,2)	439

Figure 4.2. An illustrative example of the coverage problem with three targets and its optimal solution



request r	lower left corner (x_r, y_r)	width w_r	length l_r	reward rate v_r
1	(0,7)	7	5	4
2	(1,0)	5	10	6
3	(5,5)	6	6	7
4	(4,3)	6	6	3
5	(9,2)	6	5	10

details of target		optimal position (x_t, y_t)	optimal (max) reward R
width (w_t)	length (l_t)		
4	3	(1,7)	561
		(5,5)	
		(9,4)	
		(5,8)	

Figure 4.3. An illustrative example of the coverage problem with four targets and its optimal solution

The MTP is an extension of STP (discussed in Chapter III). For applications and other related problems refer to Chapter II. The structure of MTP can be compared to the planar p-center or p-median problems which are NP-complete, if p is part of the input [25]. In fact, we conjecture that MTP is NP-complete and that it can be proved by reducing the planar p-center problem to MTP. This is a line of future research that we are working on (Chapter V). The remainder of this chapter will be as follows: We perform a theoretical study of MTP in Subsection 4.2 and prove several theoretical properties for its solution. In Subsection 4.3, we present the theoretical structure of our novel algorithm followed by an example in Subsection 4.4. This is a branch-and-bound algorithm which includes a main branch-and-bound tree along with two trees (one for each axis) to store the information required for branching in the main tree in an efficient format. Branching is done using a clustering scheme. The theoretical properties of the solution calls for a novel branching strategy for MTP. In Subsection 4.5, we present our

computational experiments on our algorithm and show that it efficiently solves large instances of the problem.

4.2. Theoretical Properties of MTP

In this subsection, we establish several theoretical properties for the solution to the MTP. We extend the concept of Critical Values (CV) (which was discussed in Subsection 3.3 for STP), to MTP and prove several results regarding the role of CVs in the solution to MTP. These results will reduce the continuous two-dimensional solution space to a set of discrete points. However we will see that the reduced space has a more complex structure and depends on the relative position of targets too.

We saw in Chapter III that in STP the CVs are generated by requests (we called them RCVs). In MTP in addition to RCVs, we have a new type of CVs that are generated by targets, which we will refer to as Target CVs or (TCVs). Let us first recall the concept of RCVs. Given a request i, four x RCVs and four y RCVs corresponding to this request are defined as follows (see Fig. 3.2):

$$x_i^1 = x_i - w_t; \quad x_i^2 = x_i; \quad x_i^3 = x_i + w_i - w_t; \; x_i^4 = x_i + w_i$$

$$y_i^1 = y_i - l_t; \quad y_i^2 = y_i; \quad y_i^3 = y_i + l_i - l_t; \quad y_i^4 = y_i + l_i$$

The RCVs are numbered from left to right from 1 to 4. We categorized RCVs into two types: The critical values are of two types: RCVs 2 and 3 are type 1 (denoted by RCV1) and RCVs 1 and 4 are type 2 (denoted by RCV2). According to Fig 3.2, $x_i^2, x_i^3, y_i^2, y_i^3$ are RCV1s and $x_i^1, x_i^4, y_i^1, y_i^4$ are RCV2s.

In MTP, each target with a fixed position defines a set of three TCVs for other targets along each axis. Fig. 4.4 shows the TCVs generated by a target 2 with a lower left corner of (x_{t_2}, y_{t_2}) for a target 1. The x and y TCVs are as follows:

$$x_{TCV}^1 = x_{t_2} - w_t; \qquad x_{TCV}^2 = x_{t_2}; \qquad x_{TCV}^3 = x_{t_2} + w_t$$

$$y_{TCV}^1 = y_1^2 - l_t = y_1^1; \qquad y_{TCV}^2 = y_1^2; \qquad y_{TCV}^3 = y_1^2 + l_t$$

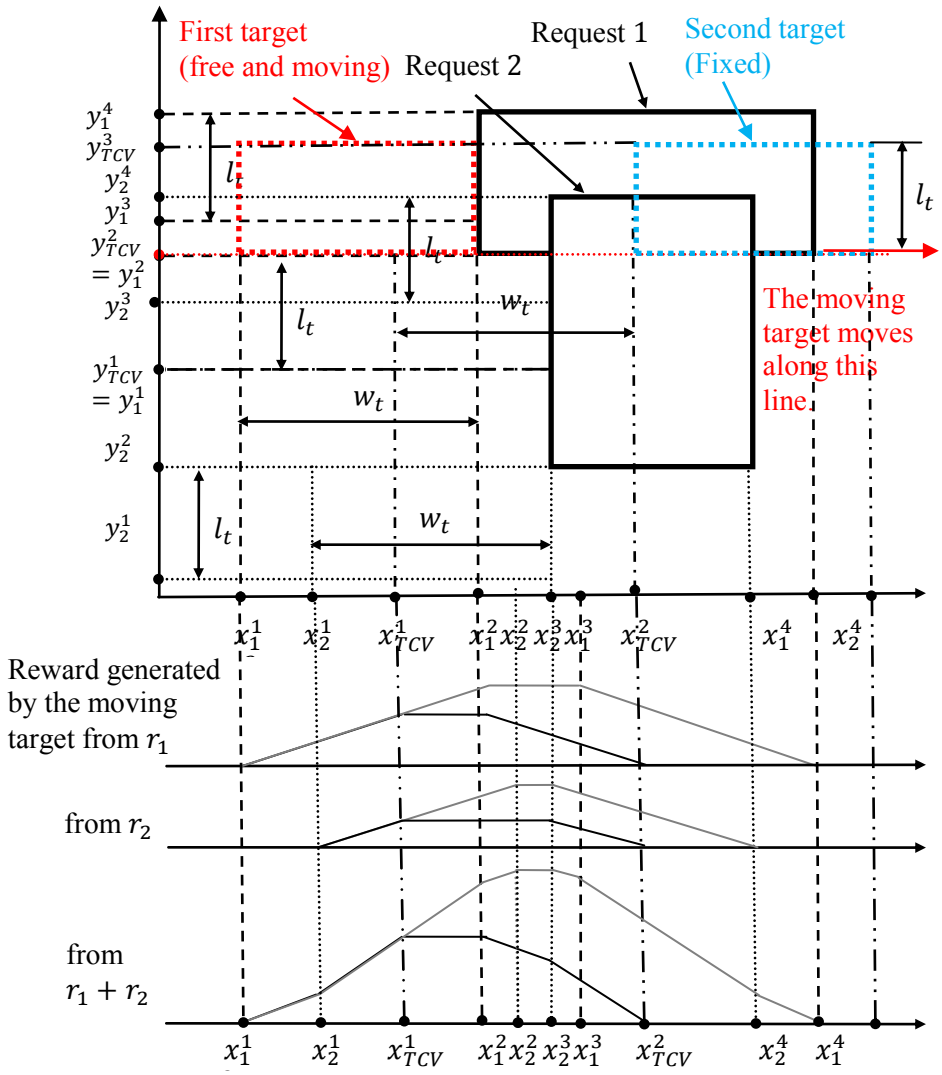


Fig. 4.4. RCVs and TCVs on x and y axes are shown for an MTP with two requests and two targets when the second target is fixed at a position. x_q^p means the pth RCV for the qth request. For any q, x_q^p is an RCV1 for $p = 2,3$, and an RCV2 for $p = 1,4$. The positions of targets are represented by their lower left corner coordinates. The lower part of the figure shows the reward (generated from each requests and total) when the first target moves parallel to x axis such that its lower left corner y coordinate is at y_1^2. Two cases are shown: when the fixed target exists (black) vs. the STP case, i.e. when the fixed target does not exist (gray). The functions are *piecewise linear* and existence of the second target changes the reward obtained from the first target

We emphasize that unlike RCVs, the TCVs are defined when the location of a target is fixed.

Assume that we move target 1 parallel to the x axis from left to right such that its lower left corner is at $y_{t_2} = y_1^2$. The graphs at the bottom of Fig, 4.4 show the reward generated by the target 1 resulting from requests 1, 2, and 1 and 2 together, respectively. The gray graphs show the reward assuming that target 2 is not present and the black graphs show the new reward functions when target 2 is fixed at the shown location. It is easy to observe that like in Fig. 3.2, here the reward graphs for target 1 are still piecewise linear functions. This is because the area covered by the target with one of its coordinates fixed is simply a linear function of the other coordinate. However the breakpoints of this function are not limited to RCVs but also include the TCVs. A similar statement is true for movement along a fixed x_{t_2} and y CVs.

Having both RCVs and TCVs we can extend the notion of *critical points* (CPs) defined in Subsection 3.3 for STP to MTP. As a result a CP for an MTP problem is a point that its x coordinate is an x RCV or TCV and its y coordinate is y RCV or TCV.

We will prove in this subsection that the solution space of MTP can be limited to a subset of CPs. First we prove a lemma that will be helpful later:

Lemma 4.3. *Consider two targets t_1 and t_2. If x_{t_1} is a TCV generated by t_2, then x_{t_2} is also a TCV generated by t_1.The same is true for y_{t_1} and y_{t_2}.*

Proof. If x_{t_1} is a TCV generated by t_2 then according to the definition of TCV in Subsection 4.2, x_{t_1} belongs to $\{x_{t_2} - w_t, x_{t_2}, x_{t_2} + w_t\}$. When $x_{t_1} = x_{t_2} - w_t$, it implies that $x_{t_2} = x_{t_1} + w_t$ which is a TCV generated by t_1. If $x_{t_1} = x_{t_2}$ then we are done. Likewise, $x_{t_1} = x_{t_2} + w_t$ implies that $x_{t_2} = x_{t_1} - w_t$ which is again a TCV generated by t_1 for t_2. Thus, we can say that if x_{t_1} is a TCV generated by t_2, then x_{t_2} is also a TCV generated by t_1. The same argument applies to TCVs along y axis. □

Theorem 4.4. *There is at least one optimal solution* $\left\{\left(x_{t_j}^*, y_{t_j}^*\right), j = 1, \dots, k\right\}$ *such that the following conditions hold:*

(a) $x_{t_p}^*$ *is an x RCV1 for at least one* $p \in \{1, \dots, k\}$ *and* $y_{t_q}^*$ *is a y RCV1 for at least one* $q \in \{1, \dots, k\}$.

(b) For every $j \in \{1, .., k\}$, $x_{t_j}^*$ *is an x RCV1 or TCV and* $y_{t_j}^*$ *is a y RCV1 or TCV.*

Proof. Assume the optimal solution is $\left\{\left(\bar{x}_{t_j}, \bar{y}_{t_j}\right), j = 1, \dots, k\right\}$. If it does not satisfy condition (a), say if there is no p for which $\bar{x}_{t_p}$ is an x RCV1, then we show that an optimal solution can be found that satisfies this condition. Note that there will not be any p for which $\bar{x}_{t_p}$ is an x RCV2 because if there is, then by moving the target parallel to x axis, either left or right, the reward increases, which is contrary to optimality of $\left\{\left(\bar{x}_{t_j}, \bar{y}_{t_j}\right), j = 1, \dots, k\right\}$. For this, in the solution $\left\{\left(\bar{x}_{t_j}, \bar{y}_{t_j}\right), j = 1, \dots, k\right\}$, move all targets together (without changing their relative positions) to the left or right such that $\bar{y}_{t_j}$ for all $j \in \{1, \dots, k\}$ is fixed (i.e. parallel to the x axis) until the first time x_{t_j} for one of the targets $j \in \{1, \dots, k\}$ becomes equal to an x RCV. The reward will not change because if the reward increases it contradicts optimality of $\{\bar{x}_{t_j}, j = 1, \dots, k\}$; if the reward decreases, since the reward function at fixed $\{\bar{y}_{t_j}, j = 1, \dots, k\}$ is piecewise linear with breakpoints at x RCV, by moving in the opposite direction, the reward increases, which is again contrary to optimality of $\left\{\bar{x}_{t_j}, j = 1, \dots, k\right\}$. Therefore the reward will remain the same and we will hit an x RCV. Notice that this RCV cannot be a RCV2 because if it is a RCV2, then by moving all targets parallel to x axis, either left or right, the reward increases, which is contrary to optimality of $\left\{\left(\bar{x}_{t_j}, \bar{y}_{t_j}\right), j = 1, \dots, k\right\}$. Therefore we have found a new optimal solution in which x_{t_p} is an x RCV1. The same argument can be used to find an optimal solution in which y_{t_q} is a y RCV1 if that is not

already satisfied. Now reassign $\left\{\left(\bar{x}_{t_j}, \bar{y}_{t_j}\right), j = 1, \dots, k\right\}$ to denote this new solution that satisfies condition (a).

If $\left\{\left(\bar{x}_{t_j}, \bar{y}_{t_j}\right), j = 1, \dots, k\right\}$ does not satisfy condition (b), then again we show that we can find another optimal solution that does. Assume $L \subset \{1, \dots, k\}$ is the target set for which $\bar{x}_{t_j}$ is neither an x RCV1 nor an x TCV for $j \in L$. Notice that $\bar{x}_{t_j}$ cannot be a RCV2 because if it is a RCV2, then by moving the target parallel to x axis either left or right, the reward increases, which is contrary to optimality of $\left\{\left(\bar{x}_{t_j}, \bar{y}_{t_j}\right), j = 1, \dots, k\right\}$. Now pick a target $l \in L$ and move it to the left (or right) with y_{t_l} fixed (i.e. parallel to the x axis) until x_{t_l} is equal to the first x RCV1 or TCV to the left (right) of $\bar{x}_{t_l}$. The reward will not change because if the reward increases it contradicts optimality of $\bar{x}_{t_l}$; if the reward decreases, since the reward function at a fixed y is piecewise linear with breakpoints at x RCV1s and TCVs, by moving in the opposite direction, the reward increases, which is again contrary to the optimality of $\bar{x}_{t_l}$. Therefore the reward will remain the same and the target will either hit an x RCV1 or TCV. If the target hits an x TCV generated by another target $l_1 \in L - \{l\}$, then according to Lemma 4.3, target l_1 also lies on an x TCV. Hence we remove target l and l_1 from the target set L, while if the target hits an x RCV, then we only remove target l. The process is repeated until $L = \emptyset$ i.e. all targets are either on an x RCV1 or TCV. The same argument is applied to optimal position of targets along y axis. Therefore there will be another optimal solution in which for every $j \in \{1, .., k\}$, $\bar{x}_{t_j}$ is an x RCV1 or TCV and $\bar{y}_{t_j}$ is a y RCV1 or TCV . □

If we define the notion of "isolated subset of targets," we can generalize part (a) of Theorem 4.4. A subset L of targets is called *an isolated subset of targets* if no target in L touches or overlaps with a target in $\{1, \dots, k\} \backslash L$.

Theorem 4.5. *There is at least one optimal solution* $\left\{\left(x_{t_j}^*, y_{t_j}^*\right), j = 1, \dots, k\right\}$ *such that every isolated subset of targets* $L \subseteq \{1, \dots, k\}$ *satisfies the following condition:* $x_{t_p}^*$ *is an x RCV1 for at least one* $p \in L$ *and* $y_{t_q}^*$ *is a y RCV1 for at least one* $q \in L$.

Proof: The proof is similar to the proof of Theorem 4.4 part (a) applied on all the isolated subsets L that do not satisfy this condition (instead of only the whole target set $\{1, \dots, k\}$). We only need to add that while doing the movement in the proof of Theorem 4.4(a), if the first time x_{t_j} for one of the targets $j \in \{1, \dots, k\}$ becomes a CV, that CV is a TCV, we stop and update the optimal solution to this solution and the set L is not an isolated subset in this new solution anymore. □

4.3. Novel Branch-and-Bound Algorithm for MTP

In this subsection we present our algorithm for solving the MTP. As a result of Theorem 4.4, we only need to search over the CPs to find an optimal solution. But observe that the CPs depend on not only RCVs but also TCVs and, as explained before, TCVs are generated by a target when its position is fixed. This causes a great deal of complication in searching the solution space. An eminently inefficient algorithm is a brute force search over all possible locations for all targets over all CPs. The reward function for every possibility is calculated and the maximum reward gives the optimal solution. This algorithm is of course of exponential complexity: The number of possibilities for positioning of k targets is roughly $O(n_r^{2k})$ and reward calculation for each takes $O(k^2 \times n_r)$ time. So total complexity of brute force algorithm is roughly $O(n_r^{2k+1} \times k^2)$.

As we conjecture MTP to be NP-hard, there is almost no chance that a polynomial algorithm to solve MTP can be devised. The algorithm we have designed in this thesis has a BB framework and is the first algorithm proposed to solve MTPs. Our algorithm follows a clever method to implicitly search the CP solution space. In this subsection we address the different components of this algorithm. If there is a higher concentration of

requests or large reward rates at some region of the plane, the chances of the optimal solution being in that region is higher. One of the main features of our BB algorithm is that it is designed to quickly concentrate on such regions to find good solutions (lower bounds). Having good lower bounds, it then avoids explicit enumeration of CPs in many other regions by calculating upper bounds on the reward if the targets are in those regions.

4.3.1. Main Body of BB

The pseudo code for the main body of our BB algorithm is shown in Fig. 4.5. Like any BB algorithm (see [4]), our algorithm is performed over a tree, which we call BBtree, and uses bounds to avoid explicitly enumerating all CPs. The nodes of this tree are called BBnodes. BBnode P in BBtree corresponds to k subsets of x CVs, called node x CV subsets and denoted by $X_P^j, j = 1, \dots, k$, and k subsets of y CVs, called node y CV sub sets and denoted by $Y_P^j, j = 1, \dots, k$. A subset X_P^j contains the candidate x CVs for target j in that BBnode. A similar statement is true for Y_P^j. Each particular subset X_P^j is either a subset of x RCV1s or a singleton x TCV. A similar statement is true for any particular subset Y_P^j. As a result, each node corresponds to k rectangular mesh where the mesh j is determined by the subsets X_P^j and Y_P^j. In Fig. 4.5, lines 1-8 initialize the BB algorithm. The input data is initialized. The BBtree is initialized by creating the root node P_0. The properties $BA(P)$ and $BT(P)$ store branching axis and branching target of each node, respectively and will be used for branching. The definition and usage of these indicators will be addressed later in Subsection 4.3.2. The function $NewChildnode(P)$ creates a child node for the node P in the BBtree with the same x and y CV subsets as those of its parent node P (if P is not Null). It also assigns the BA and BT properties of the child node the same value as $BA(P)$ and $BT(P)$, respectively. The root node P_0 has no parent so we set all the k sets $X_{P_0}^j, j = 1, \dots, k$ equal to C_x, the set of all x RCV1s, and all the k sets $Y_{P_0}^j, j = 1, \dots, k$ equal to C_y, the set of all y RCV1s.

Main:

[Initialization]

1. $Input\ set\ of\ requests\ R_0 = \{r_i[x_i, y_i, w_i, l_i, v_i],\ i = 1, \dots, n_r\};$
2. $Initialize\ P_o = NewChildNode(Null);$
3. $BT(P_0) = 1; BA(P_0) = 'X';$
4. $For\ j = 1, \dots, k$
5. $\quad X_{P_0}^j \coloneqq C_x;$
6. $\quad Y_{P_0}^j \coloneqq C_y;$
7. $End\ For$
8. $LB = InitialLowerBound(\ );$

[End of Initialization]

9. $P \coloneqq P_0;$
10. $While\ P\ \neq Null$
11. $\quad If\ UpperBound(P) \leq LB\ then$

[Upper Bound at node P is less than the best solution value known so far]

12. $\quad\quad P \coloneqq GetNextNode(P);$
13. $\quad Else\ If\ \left|X_P^j\right| = \left|Y_P^j\right| = 1\ for\ all\ j = 1, \dots, k\ then$

[All targets have fixed x, y coordinates]

14. $\quad\quad LB \coloneqq UB(P);\ Solution \coloneqq \left\{\left(X_P^j, Y_P^j\right), j = 1, \dots, k\right\};$
15. $\quad\quad P \coloneqq GetNextNode(P);$
16. $\quad Else$
17. $\quad\quad P \coloneqq Branch(P);$
18. $\quad End\ if$
19. $End\ While$
20. $Optimal\ Value = LB;$
21. $Optimal\ Targets\ Location \coloneqq Solution$

Figure 4.5. BB algorithm

First an initial lower bound (LB) on the objective is calculated. This is done using $InitialLowerBound()$ function (line 8), which is discussed in Subsection 4.3.3 Then at any given node P an upper bound is calculated for the value of the objective at that node using the function $UpperBound(P)$. This function will be addressed in detail in Subsection 4.3.3. If this upper bound is not greater than LB, the node is pruned (by bound) and we move to the next node using the $GetNextNode(P)$ function (explained

in Subsection 4.3.2). If that is not the case, then lines 13-15 handle the case where all the x and y CV subsets of the node are singletons (i.e. all targets have fixed x and y coordinates). This corresponds to a solution and in this case $UpperBound(P)$ generates the exact total reward of this solution. Since this total reward is greater than LB, LB and the best solution so far ($Solution$) are updated. The node is pruned by solution and we move to the next node using the $GetNextNode(P)$ function.

The final case is when there is no pruning by bound or solution. In this case, we branch using the *Branch(P)* function. There are several details concerning this function that are described in Subsection 4.3.2. In general, branching at any node P includes creating new nodes by clustering one particular subset out of the $2k$ subsets $X_P^j, Y_P^j, j = 1, \dots, k$, into smaller subsets but because of the existence of TCVs there are several other special cases that our $Branch(P)$ along with $GetNextNode(P)$ and $GetRightSibling(P)$ functions handle to deal with enumeration of TCVs (see Subsection 4.3.2). The algorithm terminates when all nodes are pruned and there are no more nodes to move to. $Solution$ and LB will give the optimal solution and its associated optimal reward, respectively. The pseudo codes of the functions used within the main body of BB algorithm will be explained in following subsections.

4.3.2. Branching Details

The branching is done using $Branch(P)$ as shown in Fig. 4.6. According to our node definition in Subsection 4.3.1, in our algorithm, branching of a node is performed either by decomposing (clustering) a subset of RCV1s or by generating TCVs on of the axes x or y for the position of a particular target. $BA(P)$ and $BT(P)$ are indicators denoting the axis and the target on which branching is performed at node P, respectively.

In our algorithm, we start branching at the root node by clustering $X_{P_0}^1$. That is why in Fig. 4.5 (line 3) we have $BT(P_0) = 1$ and $BA(P_0) = 'X'$ at the initialization. A depth first strategy is then used, and deeper branches are generated on X axis. This continues until X_P^1 becomes a singleton at some node. This means that target 1 now has a

fixed x position. This prompts start of branching on the next target (i.e target 2) and adds xTCVs generated by the target 1 to the set of possible locations for target 2. Therefore, the branching on target 2, in addition to the nodes with X_P^2 corresponding to clusters of x RCV1s, will also create nodes with singleton X_P^2 containing each of x TCVs generated by target 1 for target 2. This branching mechanism is repeatedly used until a node in which all targets have fixed x coordinates is reached. Then the branching on y axis starts. Lines 1-11 in Fig. 4.6, handle the branching on x axis as explained above. The comments in the pseudo code clarify the correspondence with different possible cases discussed above.

We note that the clustering of x RCV1s to populate X_P^j is performed on a clustering tree similar to what was presented in Subsection 3.3. The $GetCluster(X_P^j, 1)$ in Fig. 4.6 line 5 returns the first cluster created on the clustering tree after clustering the set of x RCVs corresponding to X_P^j. The prioritization of clusters is performed the same way as in Subsection 3.3. If the cluster has not been created in the branching of other targets, then it is created for the first time using the $Cluster(X_P^j)$ function in line 6 of Fig. 4.6 (see Fig. 3.4 for the pseudo code of this function).

Lines 10-21 of Fig, 4.6 handle the branching on y axis. An important point to keep in mind is that while branching on x axis, the sequence in which targets are selected for branching does not matter because all targets are identical. We consider an ascending order of target indexes as the sequence of selection. But while branching on y axis, each target has a fixed x coordinate associated with it. If all targets are fixed on x RCV1s, i.e. X_P^j is a singleton x RCV1 for all $j \in \{1, \dots, k\}$, then the ascending order of targets can be used for branching on y axis. But if any of the singleton X_P^j, $j \in \{1, \dots, k\}$, is a x TCV the all permutations of targets must be used on branching. This is handled in lines 13-21 of Fig. 4.6 using the set of targets with unfixed y coordinates. If a target selected for

$Branch(P)$:

1. $If\ BA(P) = 'X'\ then$ [Branching should be performed on x axis at node P]
2. $If\ BT(P) \leq k\ then$ [At least one target does not have a fixed x coordinate]
3. $C \coloneqq GetCluster\left(X_P^{BT(P)}, 1\right);$
4. $If\ C = Null\ then\ Cluster\left(X_P^{BT(P)}\right);\ End\ if$
   [Create clusters for x clustering tree]
5. $S \coloneqq NewChildNode(P);\ X_S^{BT(S)} \coloneqq C;$
6. $If\ |C| = 1\ then\ BT(S) = BT(P) + 1; End\ if$ [Select next target for branching]
7. $Else$ [All targets have fixed x coordinates at node P]
8. $S \coloneqq NewChildNode(P);\ BT(S) = 1;\ BA(S) = 'Y';$
9. $End\ if$
10. $Else\ if\ BA(P) = 'Y'\ then$ [Branching should be performed on y axis at node P]
11. $If\ all\ X_Q^j, j = 1, \dots, k\ are\ x\ RCVs\ then$
12. $Do\ lines\ 3\ to\ 6\ by\ replacing\ X\ with\ Y$
13. $Else$
14. $Unfixed: = Sorted\ list\ of\ targets\ without\ fixed\ y\ coordinate\ at\ node\ P;$
15. $If\ BT(P) = 0\ then$ [A target just got fixed y coordinate at node P]
16. $S \coloneqq NewChildNode(P);\ BT(S) = Unfixed[1];$
17. $Else$
18. $Do\ lines\ 3\ to\ 5\ by\ replacing\ X\ with\ Y;$
19. $If\ |C| = 1\ then\ BT(S) = 0; End\ if;$
20. $End\ if$
21. $End\ if$
22. $End\ if$
23. $Return\ S;$

Figure 4.6. Branching in BB algorithm

branching at node $Parent(P)$ has just got a fixed y coordinate at node P and more than one target have unfixed y coordinates (are available for branching) at node P, then we create a new child node, where a target with lowest index is selected for branching. This strategy will cause all permutations to be considered in branching.

In Fig.4.7, lines 1-9 determine traversal on BBtree using depth first strategy. $GetNextNode(P)$ finds the next node to be implemented in main function. Function $GetRightSibling(Parent(P))$ gives right sibling of node P by creating another child node for parent of P, say Q. The method to create next child of Q is as follows: Lines 13-

17 consider the case where branching is done on x axis and the set of x RCV1s for its branch correspond to its cluster. If branching target at node $Parent(Q)$ has a fixed x coordinate at node Q and branches corresponding to clusters has already been created, then TCVs are generated by fixed targets. Lines 18-23 create branch and assign a new distinct TCV generated by targets fixed along x axis at node Q, to subset X_S^j, where j is branching target at node Q. To avoid redundancy, only TCVs different from RCV1s should be considered. In lines 24-28 while branching on x axis, if all targets have a fixed x coordinate and at least one of them is an x TCV, then new child node S is created where a target, with index next to the index of branching target at node P, is selected for branching. While branching on y axis, if all targets are fixed on x RCV1s at node Q, i.e. X_Q^j is a singleton x RCV1 for all $j \in \{1, \dots, k\}$, then the next child is created either corresponding to clusters for the branching target at Q or TCVs generated by targets with fixed y coordinate at node Q. Otherwise, in lines 32-36, while branching on y axis at node Q, if branching target at $Parent(Q)$ has a fixed y coordinate at node Q then a new child node S is created, where a target with lowest index and which does not have a fixed y coordinate is selected for branching. This will cause all permutations to be considered for branching. Lines 38-45 consider the case where branching target at node Q does not have a fixed y coordinate. In this case the set of y RCV1s for its branch correspond to its cluster. If there does not exist any branching target for node $Parent(Q)$ and branches corresponding to clusters have already been created, then TCVs are generated by fixed targets. Lines 42-47 create branch and assign a new distinct TCV generated by targets fixed along x axis at node Q, to subset Y_S^j, where j is branching target at node Q. To avoid redundancy, only TCVs different from RCV1s should be considered. In this way $GetNextNode(P)$ gives next child node using $GetRightSibling(Parent(P))$ and moves up on BBtree if the child node does not exist. The idea is of this branching scheme is to quickly focus on the regions of the plane that are populated by the requests and thus have higher probability of containing the optimal location.

$GetNextNode(P)$:

1. $If\ Parent(P) = Null\ then$ [Current node is root node]
2. $\quad Return\ Null;$
3. $Else$
4. $\quad S \coloneqq GetRightSibling(P);$
5. $\quad If\ S \neq Null\ then$
6. $\quad\quad Return\ S;$ [Move Right on BBTree]
7. $\quad Else$
8. $\quad\quad Return\ GetNextNode\big(Parent(P)\big);$ [Move Up on BBTree]
9. $End\ if$

$GetRightSibling(P)$:

10. $Q \coloneqq Parent(P); S \coloneqq Null;$
11. $l = GetChildNumber(P) + 1;$
12. $If\ BA(Q) = 'X'\ then$ [Branching should be performed on x axis at node Q]
13. $\quad If\ BT(Q) \leq k\ then$ [At least one target does not have a fixed x coordinate]
14. $\quad\quad C \coloneqq GetCluster\left(X_Q^{BT(Q)}, l\right)$
15. $\quad\quad If\ C \neq Null\ then$ [There are x RCV clusters remaining for target BT(Q)]
16. $\quad\quad\quad S \coloneqq NewChildNode(Q);\ X_S^{BT(S)} \coloneqq C;$
17. $\quad\quad\quad If\ |C| = 1\ then\ BT(S) = BT(Q) + 1;\ End\ if$
18. $\quad\quad Else\ If\ BT(Q) > BT\big(Parent(Q)\big)\ then$ [If a target just got a fixed x coordinate]
19. $\quad\quad\quad C \coloneqq\ next\ xTCV\ from\ the\ set\ of\ xTCVs\ generated\ by\ targets\ in\ \{1, \dots, BT(Q) - 1\}$;
[Create x TCV corresponding to targets having fixed x coordinate]
20. $\quad\quad\quad If\ C \neq Null\ then$
21. $\quad\quad\quad\quad S \coloneqq NewChildNode(Q);\ X_S^{BT(Q)} \coloneqq C;\ BT(S) = BT(Q) + 1;$
22. $\quad\quad\quad End\ if$
23. $\quad\quad End\ if$
24. $\quad Else$ [BT(Q) = k + 1, i. e. all targets have fixed x coordinates at node Q]
25. $\quad\quad If\ any\ of\ X_Q^j, j = 1, \dots, k\ is\ an\ x\ TCV\ then$
26. $\quad\quad\quad S \coloneqq NewChildNode(Q);\ BT(S) = BT(P) + 1; BA(S) = 'Y';$
27. $\quad\quad End\ if$
28. $\quad End\ if$

Figure 4.7. Functions used in BB algorithm

```
29. Else if BA(Q) =′ Y′ then [Branching should be performed on y axis at node Q]
30.     If all X_Q^j, j = 1, … , k are x RCVs then
31.         Do lines 14 to 23 by replacing X with Y
32.     Else
33.         Fixed ≔ Sorted list of targets with fixed y coordinate at node Q;
34.         Unfixed ≔ Sorted list of targets {1, … , k}\Fixed at node Q;
35.         If BT(Q) = 0 then [A target just got fixed y coordinate at node Q]
36.             S ≔ NewChildnode(Q); BT(S) = Unfixed[l];
37.         Else
38.             C ≔ GetCluster(Y_Q^{BT(Q)}, l)
39.             If C ≠ Null then   [There are y RCV clusters remaining for target BT(Q)]
40.                 S ≔ NewChildnode(Q); Y_S^{BT(S)} ≔ C;
41.                 If |C| = 1 then BT(S) = 0; End if
42.             Else If BT(Parent(Q)) = 0 then
43.                 C ≔ next y TCV from the set of y TCVs generated by targets
                        in Fixed;
                    [Create y TCV corresponding to targets having fixed y coordinate]
44.                 If C ≠ Null then
45.                     S ≔ NewChildnode(Q); Y_S^{BT(Q)} ≔ C; BT(S): = 0;
46.                 End if
47.             End if
48.         End if
49.     End if
50. End if
51. Return S;
```

Figure 4.7. (cont')

4.3.3. Bounding

In Fig.4.8, LB denotes the lower bound on maximum reward and S_{r_i} is the maximum reward possible from covering request i by a target t if target t can be placed anywhere, i.e. $S_{r_i} = v_i \max_{(x_t, y_t)} A(r_i \cap t)$. Lines 3-7 find request m with $\max_{\mathrm{r}\in\mathrm{R}} S_r$ and trim or split (will be explained later in this subsection) the request such that new set of requests are not covered by target j placed at lower left corner of request m. The set of requests R is updated by adding new set of requests R' and removing request m. These steps are repeated for each target to find maximum reward possible from the updated set of requests and lower bound is calculated by adding them.

Function $UpperBound(P)$ calculates the reward upper bound at each node (an upper bound (UB) on the reward when the targets' position is limited to subsets of CVs corresponding to that node). The bounding function depends on the number of CVs in all $2k$ subsets at a node. When all subsets have one element, UB is calculated by trimming and splitting the requests (Fig.4.8) and otherwise, by using PVT algorithm [29]. The PVT algorithm scans the CPs in movements parallel to one of the axes and finds the reward at the next CP by calculating the slope of the piecewise linear functions. We store the reward at each CP in a two-dimensional array, denoted as *regional reward array* (RR), where each dimension corresponds to RCV1s at root node along an axis. In Fig.4.8, lines 29-37 calculate UB by summing maximum rewards among the CPs in the rectangular meshes (or RR) for all targets. If a target lies on an x TCV, say x, then we find maximum reward among columns $\underline{x}$ and $\overline{x}$ of RR. The $\underline{x}, \overline{x}$ are the greatest and smallest x RCV1, respectively, such that $\underline{x} \leq x \leq \overline{x}$. Also, if there does not exist any RCV1, $\underline{x} \leq x$ or $\overline{x} \geq x$ then we only consider column $\overline{x}$ or $\underline{x}$, respectively. Similarly, if a target lies on a y TCV then we find maximum reward among rows $\underline{y}$ and $\overline{y}$ of RR. This bound does not consider overlapping of the targets and hence gives value greater than optimal value.

Lines 11-27 trim or split requests to gives optimal value when all targets are fixed. The trimming is performed by only reducing either length or width of a request, while splitting is the division of a request region into more than one rectangular requests. Fig 4.9 gives an example of trimming and splitting of a request. In Fig. 4.9, we assume that target with dark boundaries is fixed and to find the reward from request 4, by the target with dashed boundaries, we first trim the request to a region, which we call RTrim1 and then split the trimmed area into two requests to remove region of the request already covered by the fixed target. It is important to note that request splitting (creation of additional requests) takes place if and only if a corner of fixed target lies inside the request.

InitialLower Bound ( ):

1. $R := R_0; LB = 0;$
2. $For\ j = 1, \dots, k$
3. $\quad LB = LB + max_{r \in R}\, S_r\ ;$
4. $\quad r_m = arg\, max_{r \in R}\, S_r;$
5. $\quad \left(x_{t_j}, y_{t_j}\right) = \left(x_{r_m}, y_{r_m}\right);$
6. $\quad Trim\ or\ Split\ the\ request\ m\ to\ new\ set\ of requests\ R'$
7. $\quad such\ that\ no\ point\ of\ R' is\ covered\ by\ target\ j;$
8. $\quad R = \{R \backslash r_m\} \cup R';$
9. $End\ for$

UpperBound $(\boldsymbol{P})$*:*

10. $UB = 0;$
11. $If\ \left|X_P^j\right| = \left|Y_P^j\right| = 1\ for\ all\ j = 1, \dots, k\ then$ [All targets have fixed x, y coordinates]
12. $\quad For\ j = 1, \dots, k$
13. $\quad\quad R_j = \emptyset\ ;$
14. $\quad\quad For\ r \in R_0$
15. $\quad\quad\quad Trim\ request\ r\ to\ new\ request\ r' such\ that$
16. $\quad\quad\quad every\ point\ of\ r' is\ covered\ by\ the\ target\ j\ ;$
17. $\quad\quad\quad R_j = \{R_j \backslash r\} \cup r';$
18. $\quad\quad End\ For$
19. $\quad\quad For\ r \in R_j$
20. $\quad\quad\quad For\ m = 1, \dots, j$
21. $\quad\quad\quad\quad Trim\ or\ Split\ the\ request\ r\ to\ new\ set\ of requests\ R'$
22. $\quad\quad\quad\quad such\ that\ no\ point\ of\ R' is\ covered\ by\ target\ m;$
23. $\quad\quad\quad\quad R_j = \left\{R_j \backslash r\right\} \cup R';$
24. $\quad\quad\quad End\ for$
25. $\quad\quad End\ For$
26. $\quad\quad UB = UB + \sum_{r \in R_j} S_r\ ;$
27. $\quad End\ For$
28. $Else$
29. $\quad For\ j = 1, \dots, K$
30. $\quad\quad If\ either\ X_p^j\ is\ an\ x\ TCV\ (call\ it\ x)\ then\ X = \left\{\underline{x}, \overline{x}\right\};$
    $[\underline{x}, \overline{x}$ are the greatest and smallest x RCV1,respectively, st. $\underline{x} \le x \le \overline{x}]$
31. $\quad\quad Else\ \ X = X_P^j\ ;$
32. $\quad\quad Endif$
33. $\quad\quad If\ either\ Y_p^j\ is\ a\ y\ TCV\ (call\ it\ y)\ then\ Y = \left\{\underline{y}, \overline{y}\right\};$
    $[If\ \nexists\ \underline{y}\ or\ \overline{y}\ then\ Y = \overline{y}\ or\ Y = \underline{y}\ respectively\ ]$
34. $\quad\quad Else\ \ Y = Y_P^j;$
35. $\quad\quad Endif$
36. $\quad\quad UB = UB + max_{x' \in X, y' \in Y}\, RR[x'][y']\ ;$
37. $\quad End\ For$
38. $Endif$

Figure 4.8. Bounding functions

In line 13, R_j denotes a set of trimmed requests for target j at node P. These are obtained by trimming original requests to the target region in lines 14-18. The requests are further trimmed or split (lines 19-25) to new requests such that no point of the new requests is covered by the targets already considered. The optimal value is calculated by summing S_r for all new requests. That is the how we calculate $UpperBound(P)$ in our algorithm. Calculation of lower and upper bounds typically results in eliminating many subsets of k CPs without calculating reward for them explicitly and that is the main reason that our algorithm works fast and memory efficient.

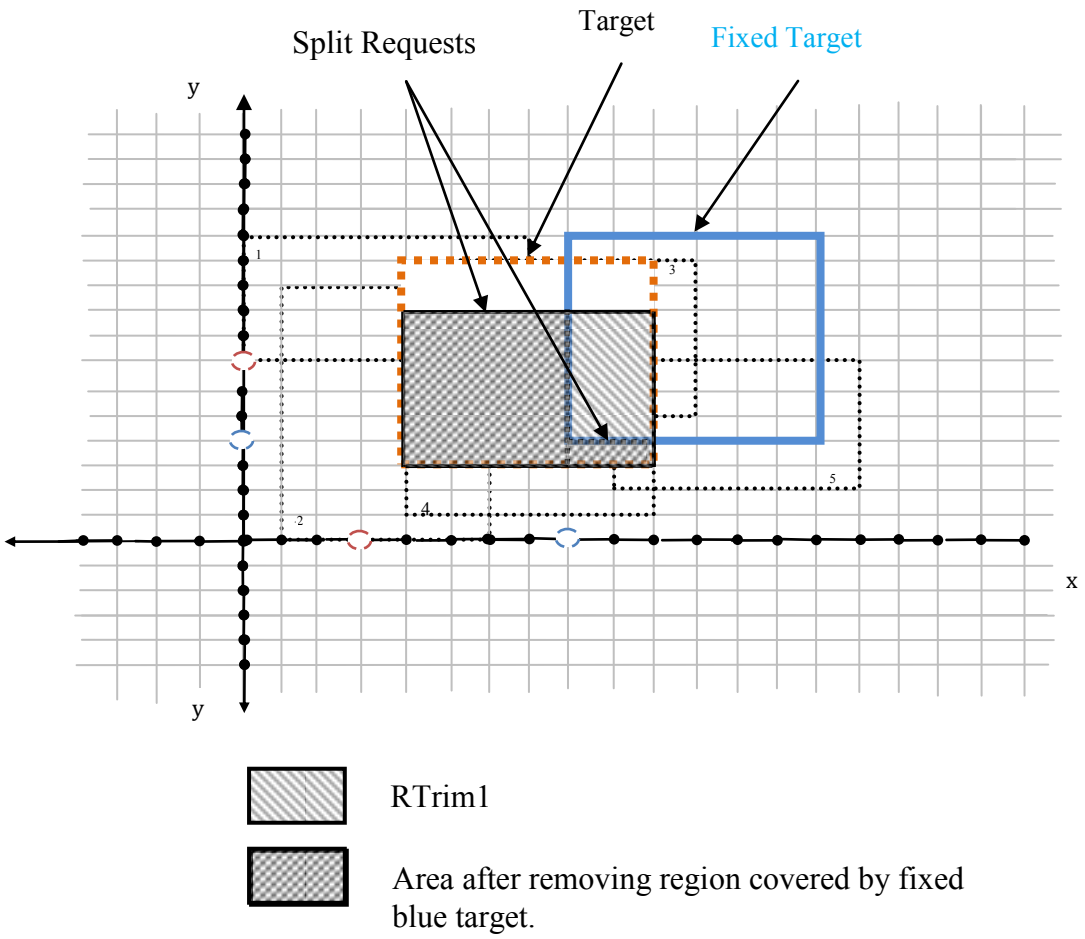


Figure 4.9. An Example of trimming and splitting of request 4

4.4. An Example for BB Algorithm

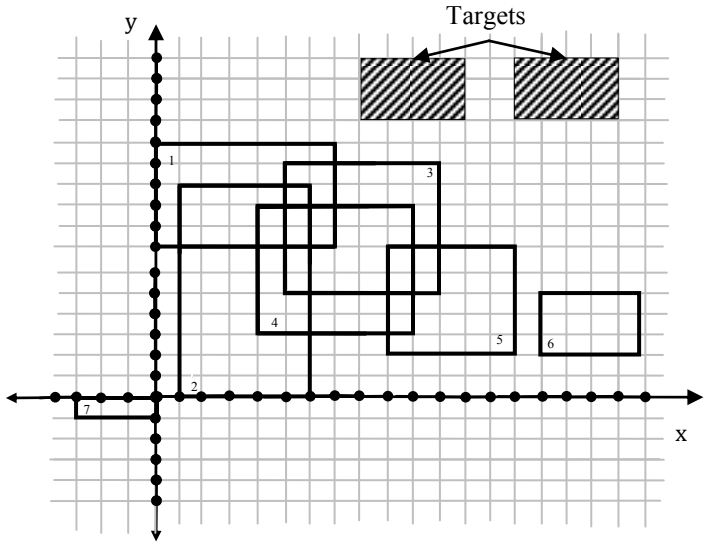
In this subsection, we introduce an example for MTP and show how our algorithm solves it. Fig. 4.10 shows the picture and data of coverage problem with seven requests and two targets. We reduce the continuous two-dimensional solution space to a set of discrete points for both targets. In Figs. 4.12 and 4.13, C_x and C_y are the initial set of RCV1s along x and y axis respectively for the problem. A set of x RCV1s $C_x^{ijk\dots lmn}$ represents n th child of a cluster of x RCV1s $C_x^{ijk\dots lm}$ in cluster tree for x axis. Similarly, a node on cluster tree for y axis can be defined. Our BB algorithm is performed over a tree which we call BBtree, as shown in Fig. 4.10 and it uses bound to avoid explicitly enumerating all points in solution space. Each of the Figs. 4.14-4.25, corresponds to a node in BBtree and in order to represent the solution space for the targets at a node, we use orange dots and blue colored circles on x and y axis. In the figure, two different shaded regions are areas which can be covered by the two target rectangles respectively.

In Fig. 4.11, we initialize BBtree by creating node A associated with C_x and C_y for each target. LB and UB denotes the lower bound and upper bound on maximum reward respectively. At node A, LB is obtained by positioning the targets at the lower left corners of request 5 and 3, using function $LowerBound(\,)$ in Fig. 4.8. The $LB(= 4 \times 3 \times (10 + 7))$ does not consider the overlapping with other requests. The UB is calculated using PVT algorithm [29] and trimming or splitting of requests (for more details refer to Subsection 4.3.3). In BBtree, a node is branched when UB is greater than LB at the node and the branches are determined either by clustering the set of x or y critical values or a singleton x or y TCV. At node A, we branch over $X_P^1(:= C_x)$ corresponding to its clusters in cluster tree for x axis. The clustering of C_x and C_y is performed as follows: In a sorted set of RCV1s, whenever the distance between two consecutive RCV1s is greater than 50% of the maximum distance between two consecutive RCV1s, we break the set at that point and create a new cluster as shown in Figs. 4.12 and 4.13. We also assign $PRIORITY$ to the clusters and consider branches in the order of priority. The priority value we use is the summation of the reward rates of

the requests whose RCV1s are in the cluster divided by number of RCV1s in cluster. Note that we create clusters in cluster tree only when they are to be required in BBtree. Also, we are using a depth first search strategy that only creates the BBnodes when they are to be considered. We branch over X_P^1 until it has only one element at node C and then branch over X_P^2. It is important to note that in Fig. 4.11, we are not showing all nodes of BBtree. The sequence of nodes not shown in BBtree is represented by a downwards dashed arrow in the figures. The branching on target 2, in addition to the nodes with X_P^2 corresponding to clusters of x RCV1s, also creates nodes I (in Fig. 4.11) with singleton X_P^2 containing each of x TCVs generated by target 1 for target 2. In order to avoid redundancy, the TCV should be different from the elements of C_x otherwise the node is pruned as shown by cross mark below node I in the figure. This branching mechanism is repeatedly used until node E in which all targets have fixed x coordinates is reached. Since at node E all targets are fixed on x RCV1s, i.e. X_P^j is a singleton x RCV1 for all $j \in \{1,2\}$, so we use the ascending order of targets for branching on y axis. But if any of the singleton X_P^j, $j \in \{1,2\}$, is a x TCV like at node J then all permutations of targets must be used on branching (as shown in Fig. 4.11). Note that in order to cover all possible cases of our algorithm in this example, we have modified few nodes in the BBtree (Fig. 4.11) and their UB and LB.

Figs. 4.14-4.25 show how the branching at a node subdivides the subspace into two or more subspaces. All nodes below node F in BBtree are branched along y axis in similar manner as branched along x axis, until both targets get fixed along y axis. At node H both targets' location are fixed (Fig. 4.21) and a better solution is found to update the LB. Since the node H is an end leafnode, so we find next node by creating right sibling of node H or next child node of $Parent(H)$ and moves up on BBtree if the child node does not exist. The algorithm terminates when there are no more nodes to consider which means after pruning node K and L, the BBtree gives LB equal to optimal solution value. Fig. 4.25 is the graphical representation of a optimal node of the BBtree. Calculation of lower and upper bounds typically results in eliminating many

combinations of the CPs in without calculating reward for them explicitly and the branching scheme enables the algorithm to quickly concentrate on regions with higher chance of containing the optimal solution.



request r	lower left corner (x_r, y_r)	width w_r	length l_r	reward rate v_r
1	(0,7)	7	5	4
2	(1,0)	5	10	6
3	(5,5)	6	6	7
4	(4,3)	6	6	3
5	(9,2)	6	5	10
6	(-3,-1)	3	1	0.5
7	(15,2)	4	3	1

target t	width w_r	length l_r
1	4	3
2	4	3

Fig. 4.10: An illustrative example of coverage problem with two targets

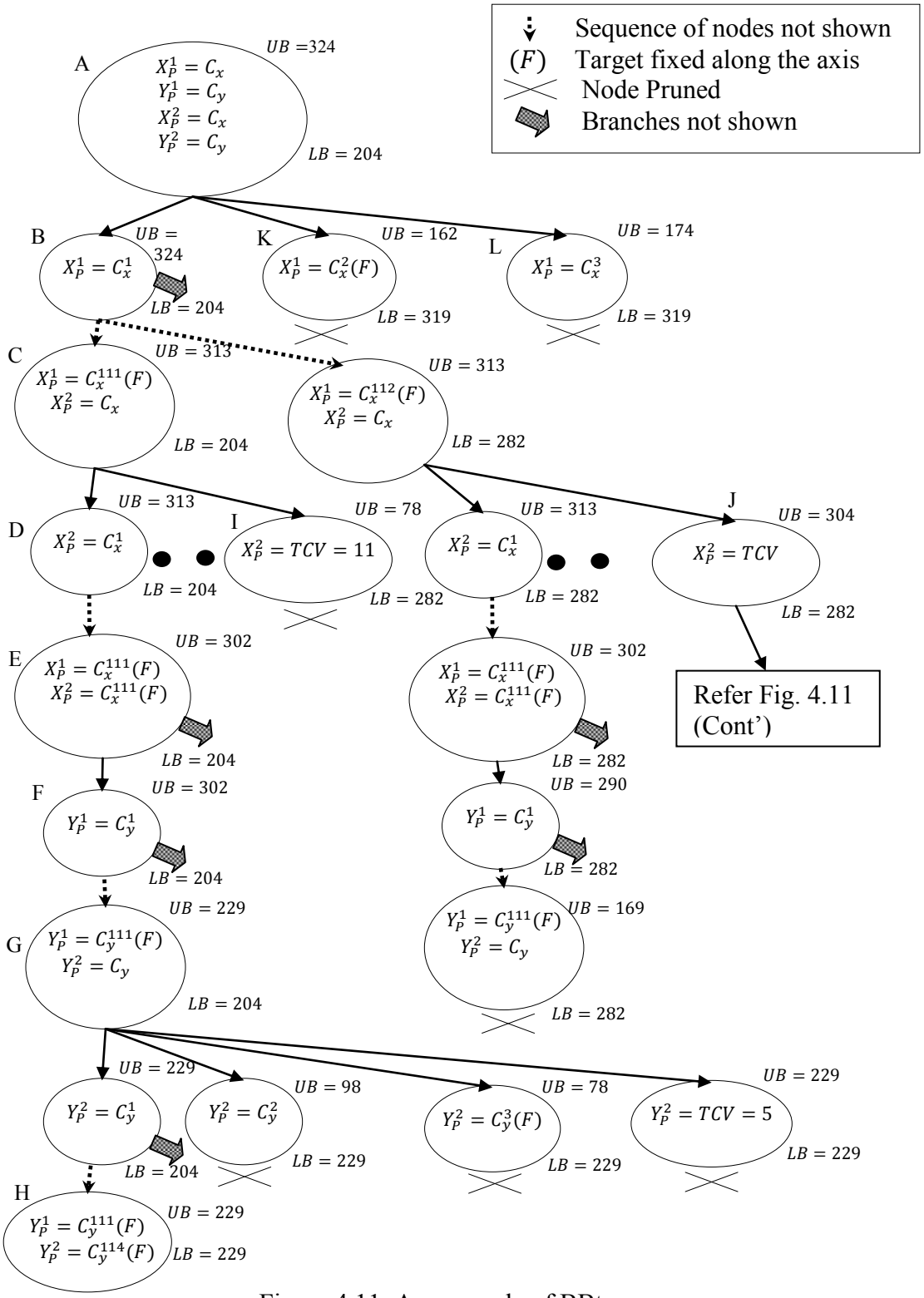


Figure 4.11. An example of BBtree

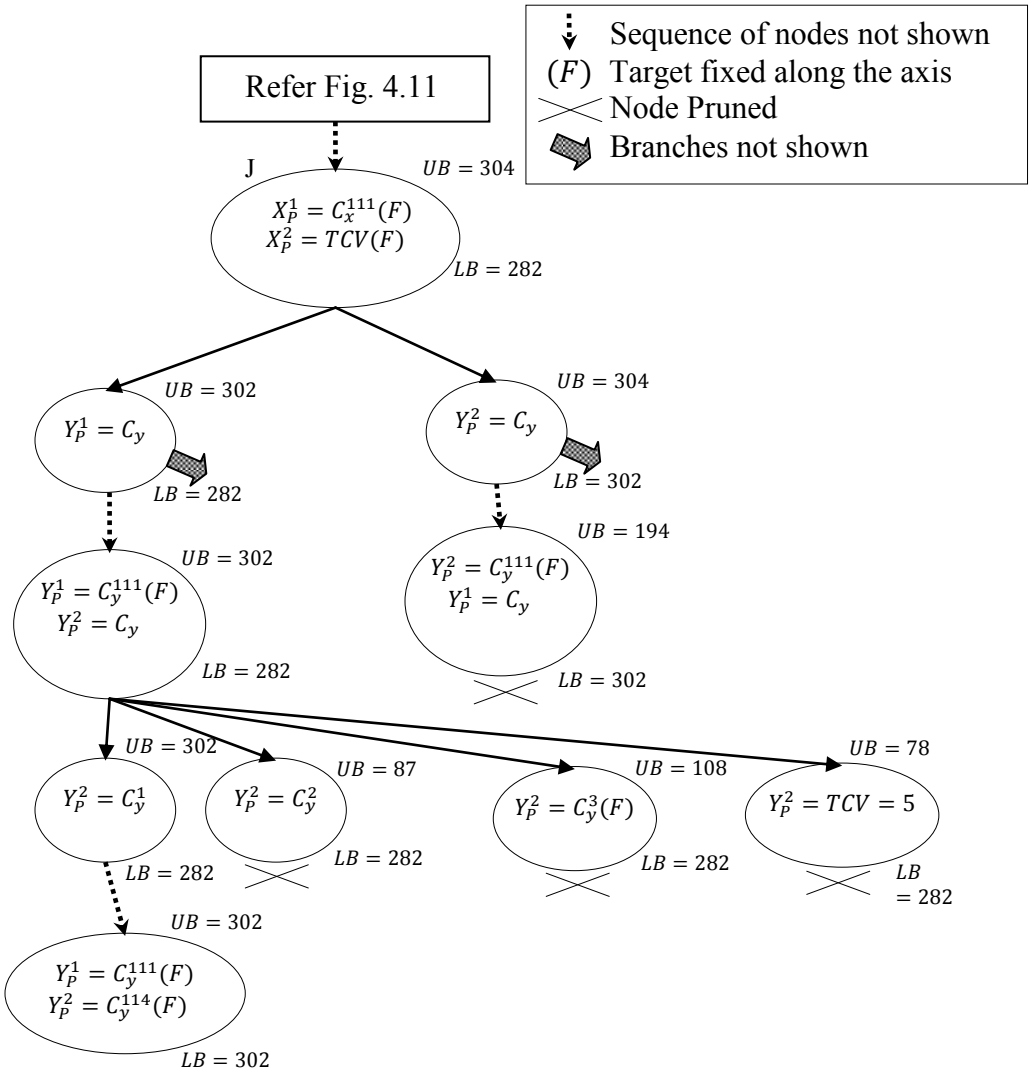


Figure 4.11. (Cont')

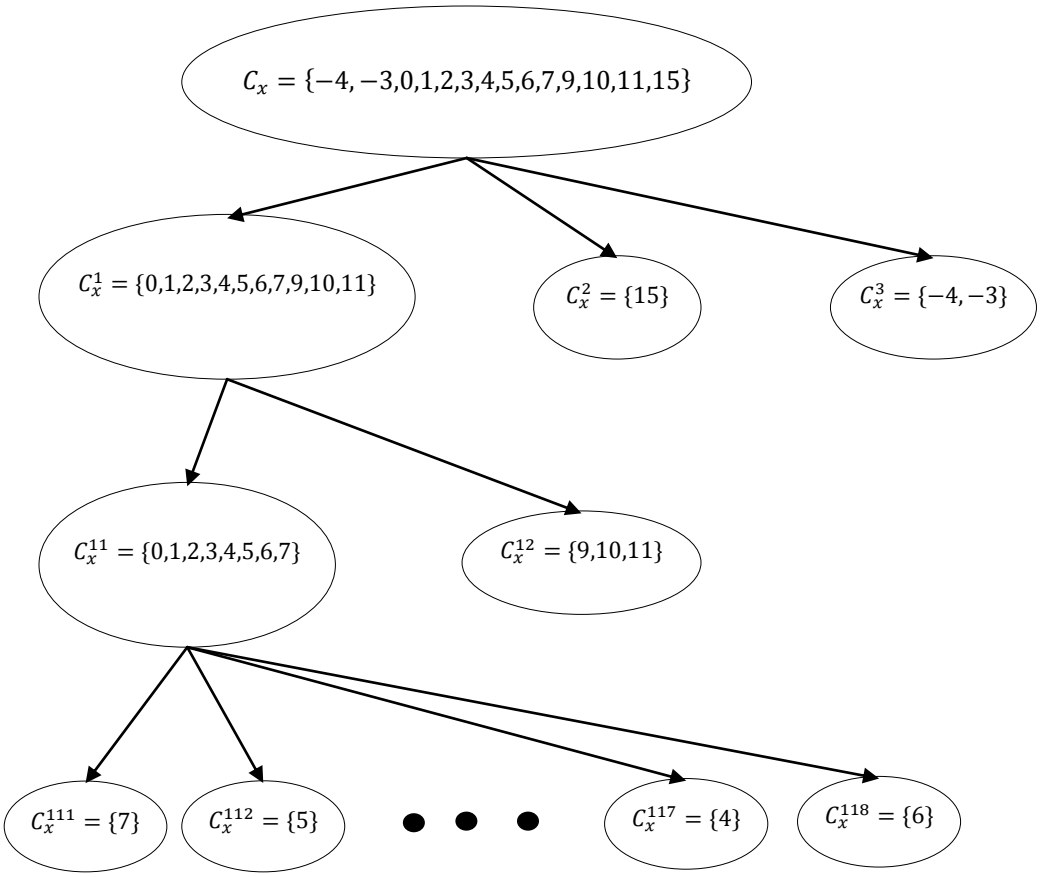


Figure 4.12 . Cluster tree for x axis

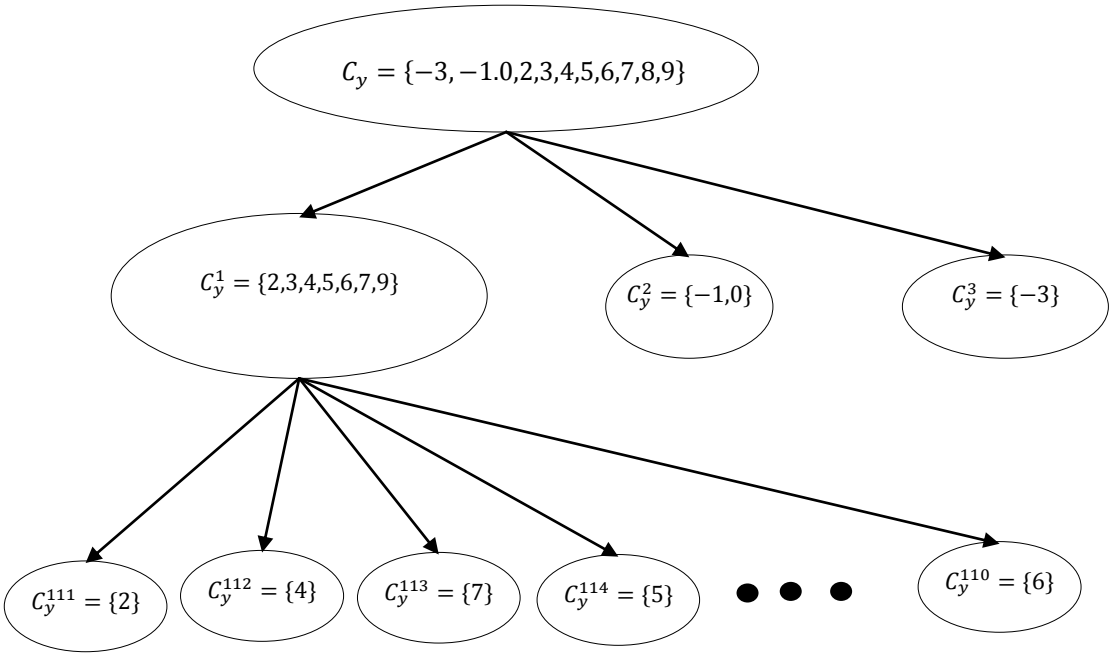


Figure 4.13. Cluster tree for y axis

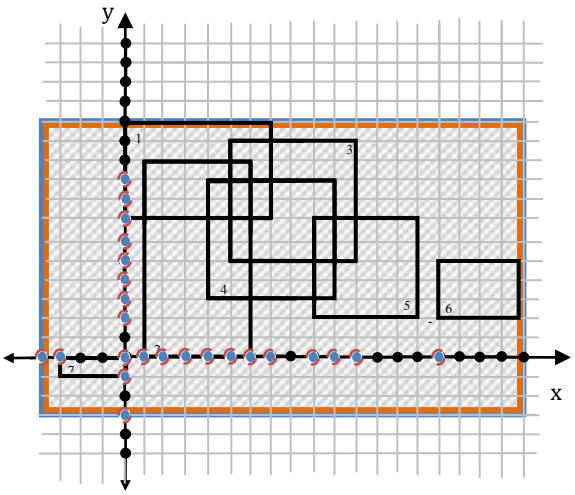


Figure 4.14. Graphical representation for node A in BBtree

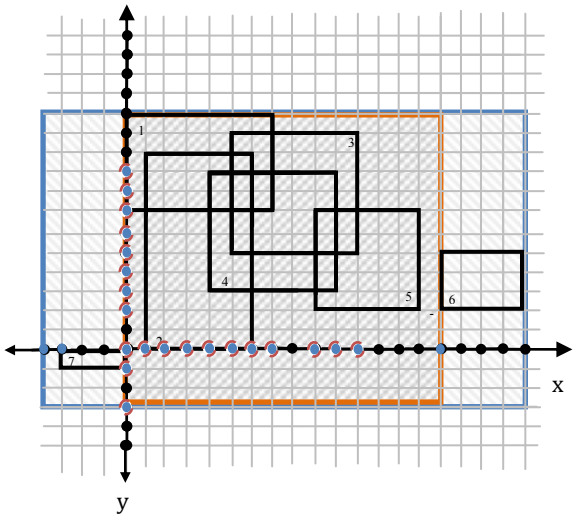


Figure 4.15. Graphical representation for node B in BBtree

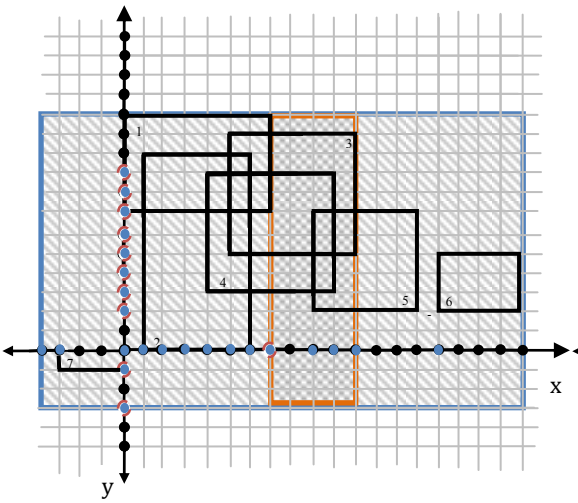


Figure 4.16. Graphical representation for node C in BBtree

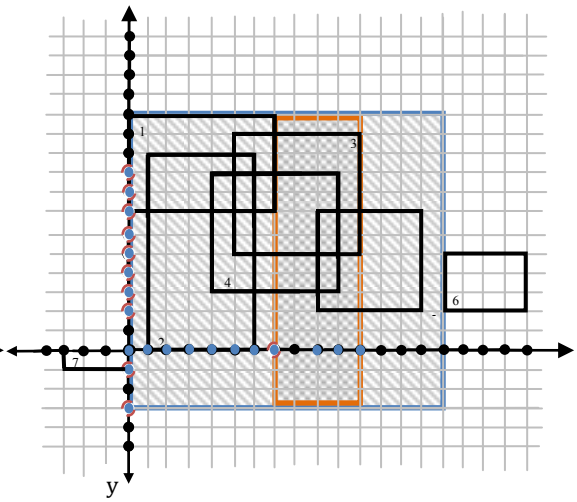


Figure 4.17. Graphical representation for node D in BBtree

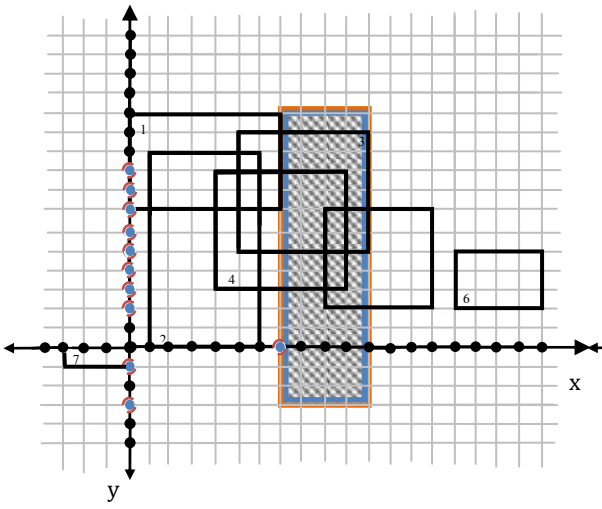


Figure 4.18. Graphical representation for node E in BBtree

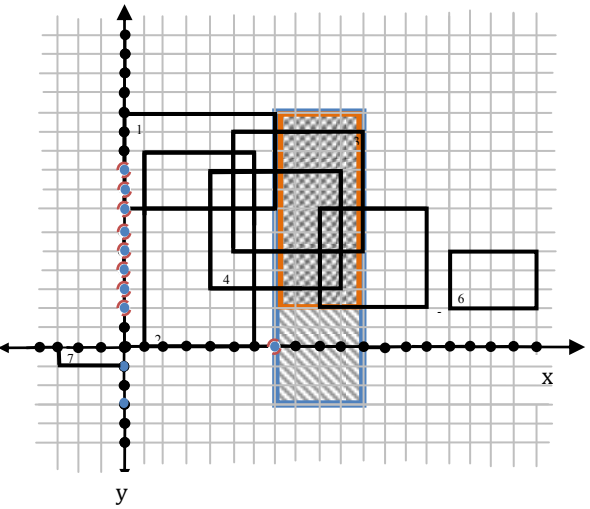


Figure 4.19. Graphical representation for node F in BBtree

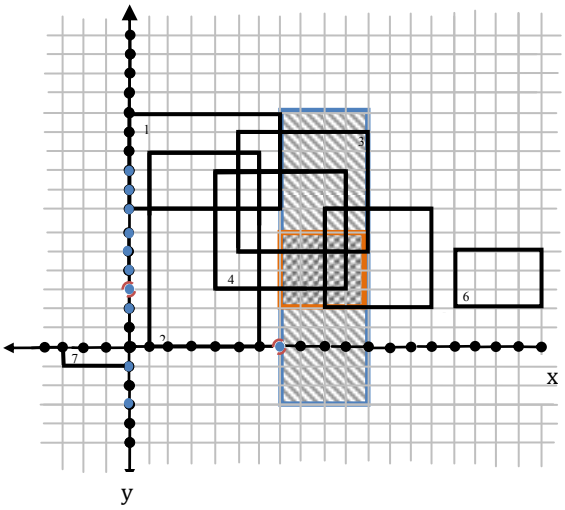


Figure 4.20. Graphical representation for node G in BBtree

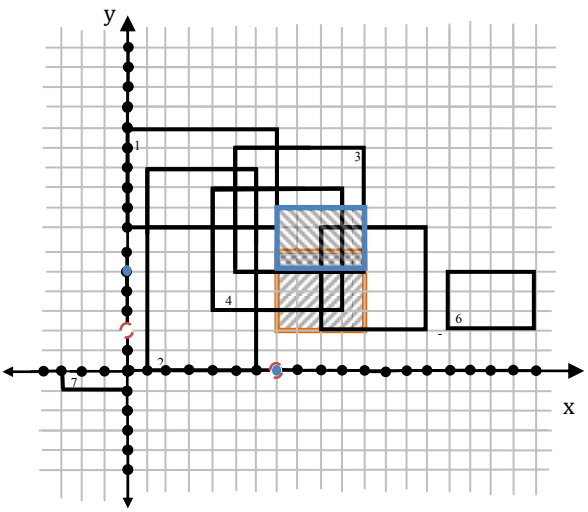


Figure 4.21. Graphical representation for node H in BBtree

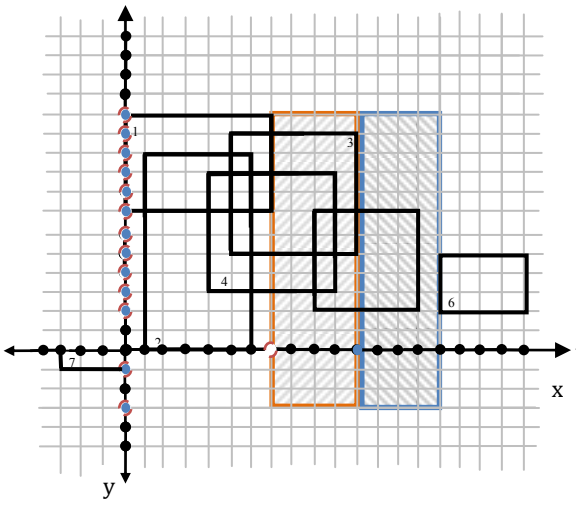


Figure 4.22. Graphical representation for node I in BBtree

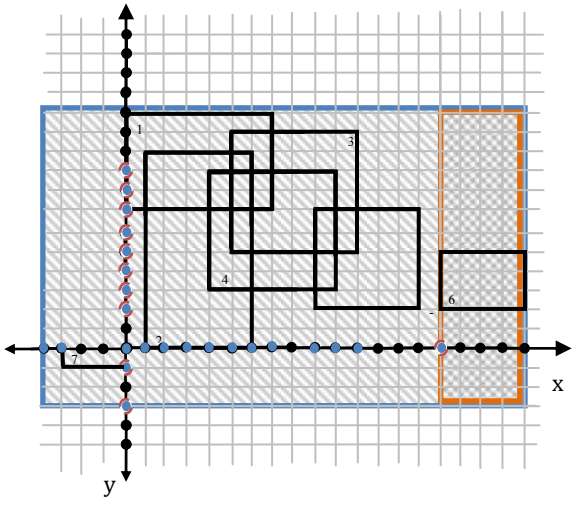


Figure 4.23. Graphical representation for node K in BBtree

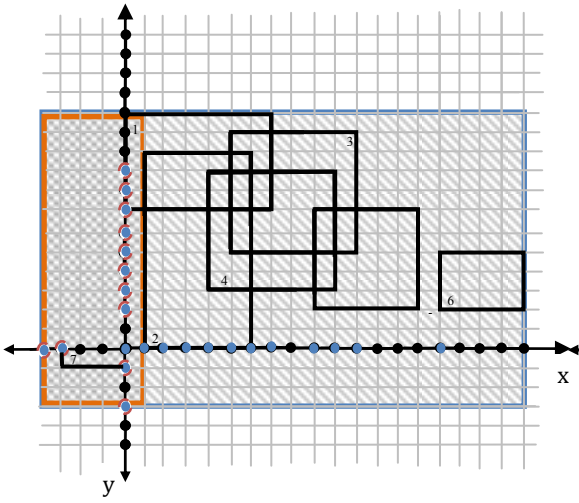


Figure 4.24. Graphical representation for node L in BBtree

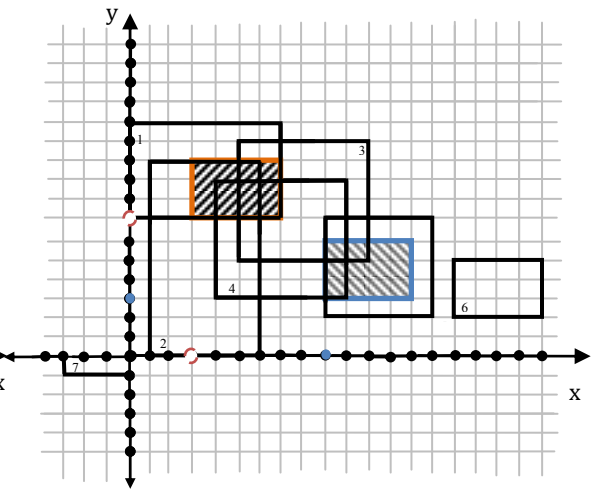


Figure 4.25. Graphical representation for Optimal Node in BBtree

4.5. Computational Results

We generated random instances with different number of requests and targets and analyze the performance of our algorithm. Similar to STP, we generated three categories of problems in terms of relative size of target and requests. They are shown in Table 4.2. In categories A, the target is smaller or equal to average of request sizes. In category B, the target size is considerably larger than the request sizes. In categories C, the target size can be smaller, equal or larger than average of request sizes. The requests are randomly distributed over a square region of determined size. The random requests are generated in two steps. First, we generate three points in the region to represent locations of interest, which we call as center points. For each center point, we use a radius of interest. Then, we generate requested viewing zones. To generate a requested viewing zone, we create six random numbers. One of them is used to determine which center point the request will be associated with. Two of them are used to generate the location of the lower left corner of the request, which is located within the corresponding radius of the associated center point. The remaining three random numbers are used to generate width, length and reward of requests. We picked the best clustering scheme out of five clustering schemes, explained in Subsection 3.3. We used Scheme 1 with $\boldsymbol{\beta = 50\%}$ in all our experiments.

Table 4.2. Problem categories

Category	w_t	l_t	w_i	l_i
A	8	6	Uniform[1,15]	Uniform[1,11]
B	8	6	Uniform[1,7]	Uniform[1,5]
C	4	3	Uniform[1,15]	Uniform[1,11]

In Experiments with 2, 3 and 4 targets, the results which are shown in Table 4.3, 4.4 and 4.5 respectively, the total region is of size (100 by 100). The reward rates of requests are from Uniform [1,10] distribution. Our BB algorithm solves relatively large instances of MTP in a short time. Using our algorithm, problems with two targets and 100 requests are solved in 3 seconds. Problems with three targets and 25 requests are solved in as few as 35 seconds, and problem with four targets and 15 requests are solved in as few as 220 seconds. For larger problems the time of our algorithm increases to more than an hour, which is still an extremely small fraction of a brute force search. Our algorithm pays off by focusing on concentrated areas and finding good lower bounds to avoid considering other areas. This is supported by the time taken by our algorithm to reach solution that is proved to be the optimal at the end, as shown in Tables 4.3, 4.4 and 4.5 and Figs. 4.25 and 4.26. In Figs. 4.25 and 4.26, T denotes the total time taken by BB algorithm to solve an instance and T_1 is the time taken by LB to reach the optimal value. It can be clearly observed in graphs between LB and time that BB algorithm quickly finds the optimal spot as T_1 is much lesser than T, in fact in many cases T_1 is 2 to 35 percent of T.

The algorithm is not only fast but it is also memory efficient. Although the number of nodes shown in the computations is very large but the maximum number of node open during execution of is very small using a depth first search strategy that only creates the nodes when they are to be considered.

Table 4.3. Experiment 1

Instances with two targets and requests distributed over a region

Category	No. of Requests	No. of Nodes	BB Algorithm	
			Time when LB reaches Optimal value T_1 (in Sec)	Run Time T (in Sec)
A	5	240	0.01	0.04
	10	539	0	0.08
	15	566	0	0.09
	20	632	0	0.10
	25	4597	0.05	0.66
	50	7511	1.25	1.37
	100	354520	7.67	92.66
B	5	1019	0.01	0.14
	10	275	0.02	0.04
	15	1196	0.14	0.17
	20	2181	0.12	0.32
	25	1839	0.21	0.30
	50	8341	1.35	1.47
	100	21336	2.6	5.05
C	5	163	0.03	0.03
	10	98	0.02	0.02
	15	354	0.02	0.06
	20	541	0.04	0.11
	25	1551	0.19	0.25
	50	4402	0.86	0.98
	100	16776	3.38	3.94

Table 4.4. Experiment 2

Instances with three targets and requests distributed over a region

Category	No. of Requests	BB Algorithm		
		No. of Nodes	Time when LB reaches Optimal value T_1 (in Sec)	Run Time T (in Sec)
A	5	12543	0.69	2.19
	10	44152	0.54	7.78
	15	82717	1.3	14.52
	20	145735	5.47	27.31
	25	933775	2.19	167.67
	50	3728494	424.884	854.49
B	5	110596	0.17	19.63
	10	31867	1.2	5.44
	15	174028	16.22	31.22
	20	496859	29.63	89.87
	25	332794	43.36	65.20
	50	8165629	1241.33	1862.97
C	5	2203	0.3	0.42
	10	998	0.04	0.21
	15	18811	0.29	3.53
	20	58915	0.56	11.65
	25	164262	11.65	34.46
	50	686382	165.131	175.84

Table 4.5. Experiment 3

Instances with four targets and requests distributed over a region

Category	No. of Requests	BB Algorithm		
		No. of Nodes	Time when LB reaches Optimal value T_1	Run Time T
A	10	4008052	102.538 s	17 min
	15	92247920	35 min	7h 35 min
	20	458965905	2 h 40 min	40 h
	25	211689650	3 h 48 min	More than 50 h
B	10	6523670	391.383 s	29 min
	15	33102537	1 h	2h 17 min
	20	119490394	17 min	8 h 30 min
	25	157509381	5 h 12 min	11 h
C	10	44217	4.45 s	11.001
	15	895430	21.302 s	219.072
	20	5242345	1 min 22 s	22 min 30 sec
	25	26312750	17 min 15 s	1 h 42 min

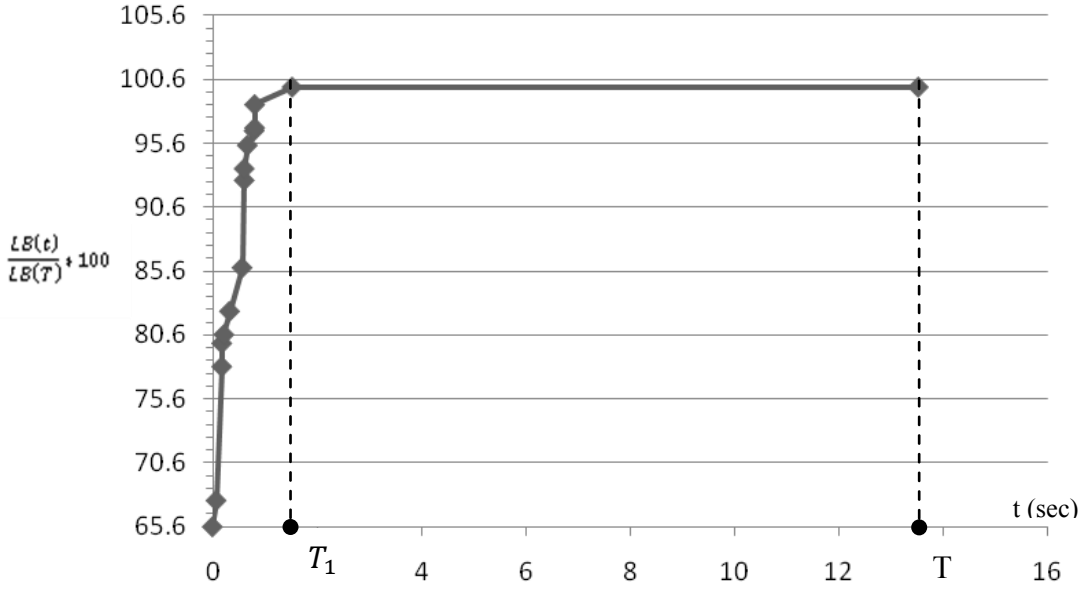


Fig. 4.26 Graph of lower bound improvement versus time for instance in Fig.4.2

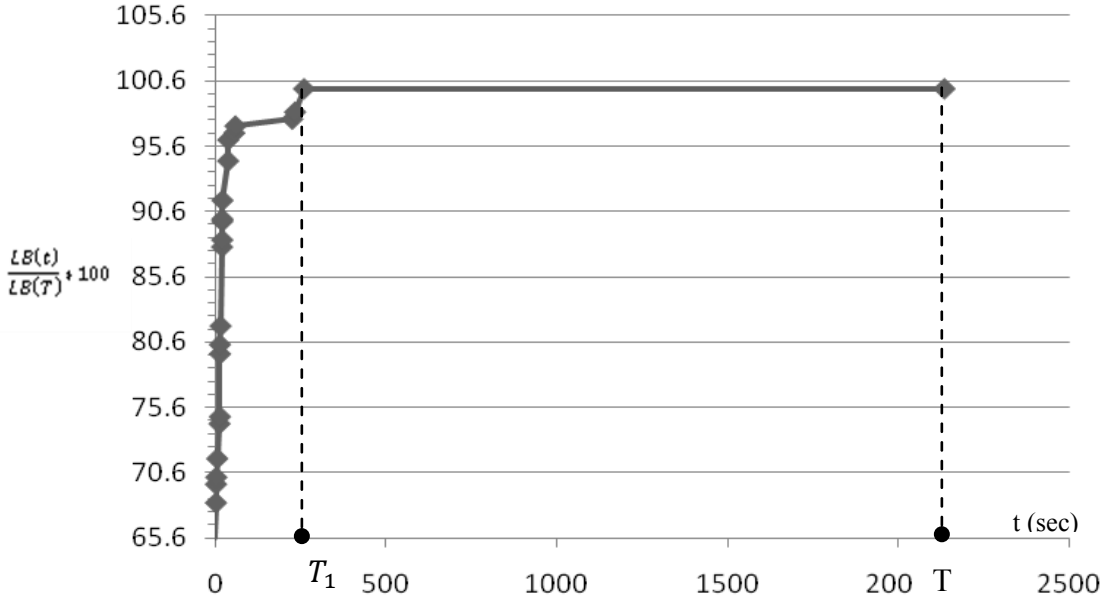


Fig. 4.27 Graph of lower bound improvement versus time for instance in Fig.4.3

CHAPTER V

CONCLUSIONS AND FUTURE WORK

5.1. Conclusions

We introduced a new class of problems, which we called Optimal Areal Positioning (OAP), and studied a special class of OAPs. In OAP, we find the optimal position of a set of floating geometric objects (targets) on two-dimensional plane to (partially) cover a set of another set of fixed geometric objects (requests) in order to maximize the total reward obtained from covered parts of requests. These problems have important applications in earth observation satellite management, tele-robotics, multi-camera control and surveillance. In this thesis, we considered a special form of OAP in which the targets and requests are parallel axes rectangles and targets are of equal size. Based on the number of targets, we classified this form into two categories: Single Target Problem (STP) and Multi-Target Problem (MTP).

- In this thesis, we developed new theoretical properties for the solution of STP and devised a new solution approach for it (refer Chapter III). This approach is based on a novel branch and bound (BB) algorithm devised over a reduced solution space. The solution space is reduced based on the theoretical properties we derived. Branching is done using a clustering scheme.

- We presented computational experiments on our algorithm for STP. The results show that in several cases our approach significantly outperforms the existing Plateau Vertex Traversal, especially for problems with many requests appearing in clusters over a large region.

- We performed a theoretical study of MTP and proved several theoretical properties for its solution (we conjecture that MTP is NP-complete and that it can be proved by reducing the planar p-center problem to MTP). We introduced a reduced solution space using these properties.

- We presented the first exact algorithm. This algorithm has a branch and bound framework. The reduced solution space is essential to our algorithm and calls for a novel branching strategy for MTP. The algorithm has a main branch-and-bound tree with a special structure along with two trees (one for each axis) to store the information required for branching in the main tree in an efficient format. Therefore it is memory and performance efficient. Branching is done using a clustering scheme. Our algorithm is capable to quickly concentrate on regions with higher chance of containing the optimal solution. Based on our literature review no work has been done so far on MTP and our theoretical results and algorithm is the first attempt to solve the problem exactly and efficiently.

- We performed computational experiments to evaluate the performance of our algorithm. Our algorithm solves relatively large instances of MTP in a short time. Using our algorithm, in average problems with two targets and 100 requests are solved in about 1 second. Problems with three targets and 25 requests are solved in about 90 seconds, and problem with four targets and 10 requests are solved in about 19 minutes. For larger problems the time of our algorithm increases to more than an hour, which is still an extremely small fraction of a brute force search. The algorithm is not only fast but it is also memory efficient. Although the number of nodes shown in the computations is very large but the maximum number of node open during execution of is very small using a depth first search strategy that only creates the nodes when they are to be considered.

5.2. Future Work

Several future research paths can be followed based on the problems and algorithms developed in this thesis:

- **NP Completeness:** In this thesis, we have conjectured that MTP is NP-complete and that it can be proved by reducing the planar p-center problem to MTP. This is a line of future research that we are working on.

- **Clustering Scheme and Bounds:** In the thesis, we picked the best clustering scheme out of five clustering schemes for branching (see Subsection 3.3). New clustering schemes can be introduced to make the algorithm more intelligent and faster. The idea of these schemes would be to more quickly focus on the regions of the plane that are populated by the requests and thus have higher probability of containing the optimal location. Our computations show that total time taken to complete the BB algorithm is more than three times the time required for the BB to reach the solution that is proved to be optimal at the end. This means that if stronger upper bounds can be developed we may be able to reduce the solution time by orders of magnitude.

- **New Forms of OAP:** We can consider various new forms of OAP in which targets and requests can be any geometric objects on the two-dimensional plane or targets can be of unequal sizes. Also, the predetermined reward associated with covering an area unit of each request can be a function of resolution. This can bring in the scaling of the target as an additional decision variable. Note that the BB algorithm can easily solve the problems with discrete resolution.

- **OAP in higher dimensions:** We can even extend OAP to higher dimensions. Immediate extensions can be 3D-STP and 3D-MTP in which targets and rectangles are parallel axes cubes.

REFERENCES

[1] Agarwal, P.K., Sharir, M.: Efficient algorithms for geometric optimization. ACM Computing Surveys 30, 412-458 (1998)

[2] Agnetis, A., Grande, E., Mirchandani, P.B., Pacifici, A.: Covering a line segment with variable radius discs. Computers and Operations Research 36(5),1423–1436 (2009)

[3] Ahn, H.K., Bae, S.W.: Covering a point set by two disjoint rectangles. Algorithms and Computation. In: Proceedings of the 19th International ISAAC Symposium, Gold Coast, Australia, 728-39 (2008)

[4] Bansal, M., Kianfar, K.: Optimal positioning of multiple rectangular targets to cover rectangular request areas. working paper (2010)

[5] Bansal, M., Kianfar, K.: An exact algorithm for coverage problem with a single rectangle. In: Proceedings of the IIE Annual Meeting, Industrial Engineering Research Conference, Cancun (2010)

[6] Chan, T.Y.: More planar two-center algorithms. Computational Geometry: Theory and Applications 13,189-198 (1999)

[7] Chvátal V.: A combinatorial theorem in plane geometry. Journal of Combinatorial Theory 18,39–41 (1975)

[8] Drezner, Z.: The p-center problem: – Heuristic and optimal algorithms. Journal of the Operational Research Society 35, 741–748 (1984)

[9] Drezner, Z.: On the rectangular p-center problem. Naval Research Logistics Quarterly 34, 229–234 (1987)

[10] Drezner, Z., Erkut, E.: Solving the continuous p-dispersion problem using non-linear programming. Journal of the Operational Research Society 36, 516–520 (1995)

[11] Eppstein, D.: Fast construction of planar two-centers. In Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms, Miami 131–138 (1997)

[12] Gabrel, V.: Improved linear programming bounds via column generation for daily scheduling of earth observation satellite. LIPN, Univ. 13 Paris, Nord, France, Technical Report (1999)

[13] Geoffrion, A.M.: Lagrangian relaxation for integer programming. Mathematical Programming Study 2, 82–114 (1974)

[14] Hochbaum DS, Maas, W.: Approximation schemes for covering and packing problems in image processing and VLSI. Journal of the ACM 32(1), 130–6 (1985)

[15] Hall, N.G., Magazine, M.J.: Maximizing the value of a space mission. European Journal of Operations Research 78, 224–241 (1994)

[16] Halperin, D., Sharir, M., Goldberg, K.: The 2-center problem with obstacles. Journal of Algorithms 42, 109–134 (Jan. 2002)

[17] Handler, Mirchandani, P. B.: Location on Networks. The M.I.T. Press, Cambridge, Massachusetts (1979)

[18] Handler, G.Y.: p-Center problems. In: Mirchandani, P.B., Francis, R.L. (eds.) Discrete Location Theory. Wiley Inter-Science, New York, 305–347 (1990)

[19] Heinrich-Litan, L., Lbbecke, M.E.: Rectangle Covers Revisited Computationally. In: Proceedings of the Fourth International Workshop on Efficient and Experimental Algorithms (WEA 05), LNCS 3503, 55-66. Springer-Verlag (2005)

[20] Huang, C.F., Tseng, Y.C.: A survey of solutions to the coverage problems in wireless sensor networks. Journal of Internet Technology 6, 18 (2005)

[21] Israeli, A., Sharon, O.: An approximation algorithm for sequential rectangle placement. Information Processing Letters 108, 407-411 (2008)

[22] Land, A.H., Doig, A.G.: An automatic method for solving discrete programming problems. Econometrica 28, 497–520 (1960)

[23] Lemaitre, M., Verfaillie, G., Jouhaud, F., Lachiver, J.M., Bataille, N.: Selecting and scheduling observations of agile satellites. Aerospace Science Technology 6, 367–381, (2002)

[24] Lu, M., Varman, P.: Optimal algorithms for rectangle problems on a mesh-connected computer. Journal of Parallel and Distributed Computing 5(2), 154-171 (1988)

[25] Megiddo, N., Supowit, K.J.: On the complexity of some common geometric location problems. SIAM Journal on Computing 13, 182–196 (1984)

[26] Nemhauser, G. L., Wolsey, L.: A. Integer and Combinatorial Optimization. Wiley-Interscience, New York, USA (1988)

[27] Saha, C., Das, S.: Covering a set of points in a plane using two parallel rectangles. In: Proc. of the 17th International Conference on Computing: Theory and Applications (ICCTA), 214-218 (2007)

[28] Sharir, M.: A near-linear algorithm for the planar 2-center problem. Discrete Computational Geometry 18, 125-134 (1997)

[29] Song, D., van der Stappen, A.F., Goldberg, K.: Exact algorithms for single frame selection on multi-axis satellites. IEEE Transactions on Automation Science and Engineering 3(1), 16-28 (2006)

[30] Song, D., Goldberg, K.: Approximate Algorithms for a Collaboratively Controlled Robotic Camera. IEEE Transactions on Robotics 23(5), 1061-1070 (2007)

[31] Tansel, B.C., Francis, R.L., Lowe, T.J.: Duality: covering and constraining p center problems on trees. In: Mirchandani, P.B., Francis, R.L. (eds.), Discrete Location Theory, John Wiley & Sons, New York (1990)

[32] Thai, M.T., Wang, F., Du, H., Jia, X.: Coverage Problems in Wireless Sensor Networks: Designs and Analysis. International Journal of Sensor Networks (special issue on Coverage Problems in Sensor Networks) 3, 191-200 (2008)

[33] Vasquez, M., Hao, J.K.: A logic-constraint knapsack formulation of a tabu algorithm for the daily photograph scheduling of an earth observation satellite. Journal of Computational Optimization and Applications 20(2), 137–157 (2001)

[34] Vasquez, M., Hao, J.K.: Upper bounds for the SPOT5 daily photograph scheduling problem. Journal of Combinatorial Optimization 7(1), 87-103 (2002)

[35] Vijay, J.: An algorithm for the p-center problem in the plane. Transportation Science 19, 235–245 (1985)

[36] Wolsey, L. A. Integer Programming. Wiley, New York, USA (1998)

[37] Xu, Y., Song, D., Yi, J., van der Stappen, F.: An approximation algorithm for the least overlapping p-frame problem with non-partial coverage for networked robotic cameras. In: IEEE international conference on robotics and automation (ICRA), Pasadena, CA, 1011–1016 (2008)

[38] Xu, Y., Song, D.: Systems and algorithms for autonomously simultaneous observation of multiple objects using robotic ptz cameras assisted by a wide-angle camera. In: International conference on intelligent robots and systems (IROS), St. Louis, USA (2009)

[39] Xu, Y., Song, D.: Systems and algorithms for autonomous and scalable crowd surveillance using robotic ptz cameras assisted by a wide-angle camera. Autonomous Robots 29, 53-66 (2010)

[40] Xu, Y., Song, D., Yi, J.: Exact Algorithms for Non-Overlapping 2-Frame Problem with Non-Partial Coverage for Networked Robotic Cameras. In: the 6th annual IEEE Conference on Automation Science and Engineering (CASE 2010), Toronto, Ontario, Canada (2010)

VITA

Name:	Manish Bansal
Address:	C/O Texas A&M University, Industrial and Systems Engineering, Zachry Bldg. College Station, TX-77843-3131
Permanent Address:	C/O Sh. Y.P. Bansal, 16 Ashoka Colony, Karnal, India-132001
Email Address:	mb.tamu@gmail.com
Education:	B.Tech., Electrical Engineering, National Institute of Technology, Kurukshetra, India, 05/2007 M.S., Industrial and Systems Engineering, Texas A&M University, College Station, 12/2010