On Piercing Set of Axis-Parallel Rectangles in 2D

Submitted by:
Anirban Ghosh

Under the Supervision of
Prof. Subhas Chandra Nandy

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF TECHNOLOGY IN COMPUTER SCIENCE

Indian Statistical Institute
203, Barrackpore Trunk Road
Kolkata 700108

CERTIFICATE

I certify that I have read the thesis prepared under my guidance by Anirban Ghosh, entitled **"On Piercing Set of Axis-Parallel Rectangles in 2D"** and in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Master of Technology in Computer Science of Indian Statistical Institute.

Prof. Subhas Chandra Nandy
Advanced Computing and Microelectronic Unit
Indian Statistical Institute

Abstract

Piercing problems often arises in facility location, and is a well-studied area of research in computational geometry. The specific piercing problem discussed in this dissertation asks for the minimum number of points required to stab a set of possibly overlapping rectangles. In other words, to determine the minimum number of facilities and their positions such that each rectangular demand region contains at least one facility located inside it. It is already proved that even if all regions are uniform sized squares, the problem is NP-hard. Therefore we concentrate on designing efficient heuristic algorithms for solving this problem. In this dissertation we do experimental studies of the piercing problem on randomly generated axis-parallel rectangles. We have implemented two approaches for piercing on random rectangles generated in a random manner (i) Greedy approach (ii) Divide-and-Conquer approach and compared the two results. We have also studied the computation of maximum independent set and minimum clique cover problems and plotted the ratio of minimum clique cover and maximum independent set for a set of n rectangles for different values of n. It is observed that Greedy Clique Cover is a 2 factor approximation of the Maximum Independent Set problem for randomly generated rectangles.

Acknowledgements

I am deeply grateful to my thesis supervisor Professor Subhas Chandra Nandy for guiding me in my research. His insightful comments and suggestions have helped me immensely in writing this thesis. I am very much grateful to Mr. Sasanka Roy who has helped me sincerely towards the realization my thesis. Besides I am also grateful to the readers of my thesis for their valuable opinions.

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

1.1 Introduction

Computational Geometry is a fascinating branch of computer science devoted to the study of algorithms for solving geometric optimization and search problems. The main impetus for the development of computational geometry as a discipline was progress in computer graphics and computer-aided design and manufacturing (CAD/CAM), .Other important applications of computational geometry include robotics (motion planning and visibility problems), geographic information systems (GIS) (geometrical location and search, route planning), integrated circuit design (IC geometry design and verification), computer-aided engineering (CAE) (programming of numerically controlled (NC) machines).

1.2 The Rectangle Intersection Graph

Geometric intersection graphs are intensively studied in the literature. A geometric intersection graph $G(V, E)$ is defined with a set of geometric objects. Usually, the vertices correspond to the geometric objects; an edge $e_{ij} \in E$ between a pair of nodes $v_i, v_j \in V$ implies that the objects corresponding to v_i and v_j intersect. The problems on geometric intersection graphs are studied for their interesting theoretical properties, and for the practical motivations. Many such graph classes allow elegant characterizations. Different graph-theoretic optimization problems, which are usually NP-hard for arbitrary graphs, can be solved in polynomial time for some geometric intersection graphs. There are many optimization problems which remain NP-hard for the geometric intersection graphs also. The following figure shows an example of rectangle intersection graph.

Any graph $G = (V, E)$ can be represented as the intersection graph of a set of axis-parallel boxes in some dimension. The boxicity of a graph with n nodes is the minimum dimension d such that the given graph can be represented as an intersection graph of n axis parallel boxes in dimension d.

A graph has boxicity at most one if and only if it is an interval graph. Every outerplanar graph has boxicity at most two, and every planar graph has boxicity at most three. If a bipartite graph has boxicity two, it can be represented as an

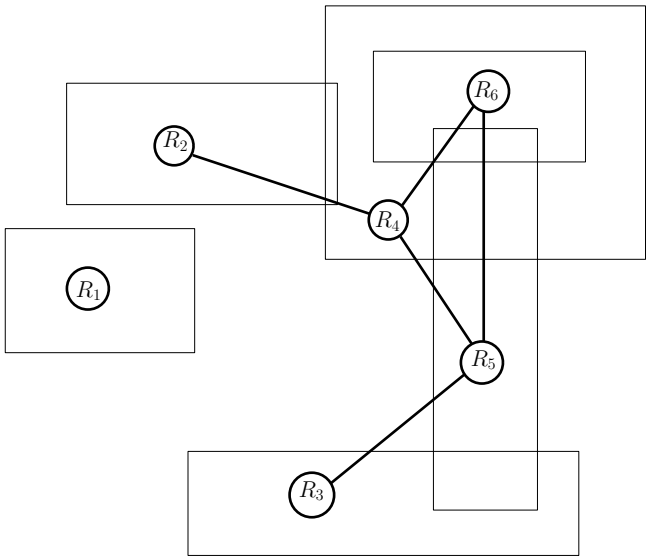

Figure 1.1: An example of Rectangle Intersection Graph

intersection graph of axis-parallel line segments in the plane. Given an arbitrary undirected graph G, testing whether the boxicity of G is a given constant k is NP-complete even if $k = 2$.

The intersection of a pair of objects is defined depending on the problem specification. For example, sometimes proper containment is considered to be an intersection and sometimes it is not. Here two types of problems are usually considered: (i) characterization problems, and (ii) solving some useful optimization problems. In the characterization problem, given an arbitrary graph, one needs to check whether it belongs to the intersection graph of a desired type of objects. The second kind of problem deals with designing efficient algorithms for solving some useful optimization problems for an intersection graph of a known type of objects. It needs to be mentioned that several practically useful optimization problems, for example, finding the largest clique, minimum vertex cover, maximum independent set, etc. are NP-hard for general graph. There are some problems for which getting an efficient approximation algorithm with good approximation factor is also very difficult. In this area of research, the geometric properties of the intersecting objects are used to design efficient algorithms for these optimization problems. The characterization problem is important in the sense that for the intersection graph of some types of objects, efficient algorithms are sometimes already available for solving the desired optimization problem.

Let us first consider the interval graph, which is the simplest type of geometric intersection graph. This is obtained by the intersection of a set of intervals on a real line. The characterization problem for the interval graph can easily be solved in $O(|V| + |E|)$ time by showing that the graph is chordal and its complementary graph is a comparability graph. All the standard graph-theoretic

optimization prob- lems, for example, finding minimum vertex cover, maximum independent set, largest clique, minimum clique cover, minimum coloring, etc, can be solved in polynomial time for the interval graph.

1.3 Geometric Packing

The general geometric packing problem asks for finding the largest sub-collection of objects among a given collection of objects in $\mathbb{R}^d$ such that no two objects in the sub-collection intersect. The number of objects in the largest sub-collection is called the *packing number*. In general the piercing number is at least as large as the packing number and in many cases approximation results for the packing problem carries over to the piercing problem. The packing problem may also be viewed from an alternative point of view: given a collection of place-holders of known geometric shapes and a corresponding collection of objects that fit the placeholders, we want to place as many objects as possible in appropriate placeholders in a non-intersecting fashion. A typical application conforming to this interpretation is found in map labeling, where non-intersecting rectangular labels are to be placed on a map with fixed points so that each point becomes a corner of one of the labels. This is a special case of the planar geometric packing problem involving rectangles. The general planar geometric packing problem allows arbitrarily shaped objects in 2D space $\mathbb{R}^2$ .The following figure shows an example of Planar Rectangle Packing where the dotted rectangles refers to the rectangle set in which no two rectangle intersect.

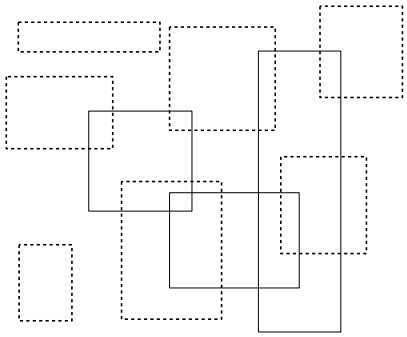

Figure 1.2: An example of the Planar Rectangle Packing

The combinatorial analogue for geometric packing is the set packing problem which is defined in the same manner: given a collection of sets, find a sub-collection with the maximum number of sets such that the sets in the sub-collection are pairwise disjoint. In the graph-theoretic formulation, the geometric packing problem actually seeks the maximum independent set in the intersection graph. The maximum independent set problem asks for a subset, of maximum cardinality, of vertices such that no two vertices in the set are adjacent. In this thesis. we will consider the maximum independent set of rectangles. The problems is NP-Hard. Thus the main focus here is getting a good polynomial time approximation algorithm for this problem. If the rectangles are arbitrary, then $O(\log \log n)$ factor approximation algorithm is recently

proposed by Parinya Chalermsook et al. If the rectangles are of unit height a 2-approximation algorithm runs in $O(n \log n)$ time.

1.4 Geometric Covering

For a collection of n given points in d-dimensional space $\mathbb{R}^d$, and the description of an object R, the general geometric covering problem asks for a set of those copies of R such that each point is contained in at least one object, and the cardinality of the set is minimum. For $d = 2$, the problem is known as *planar geometric covering problem*. The covering problem is related to both packing and piercing problems. In a covering problem, the objects usually are of similar shape or property, e.g., all objects are rectangles or all objects are disks. The objects can be non-convex as well. The following shows an instance of planar geometric covering with squares. The lines indicate square boundaries used for the cover.

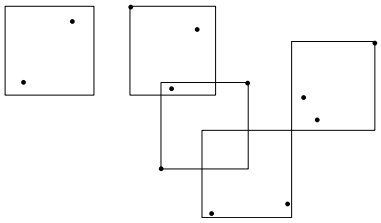

Figure 1.3: Covering with squares in 2 dimensions

Geometric covering is analogous to the well known set cover problem : given a finite set S and a collection C of subsets of S such that each element of S is contained in at least one of the subsets in C, we have to choose the minimum number of subsets in C such that each element of S is contained in at least one of the chosen subsets. A graph representing an instance the geometric covering problem is different from an intersection graph. In such a representation a vertex usually represents a point and two vertices are considered adjacent if they can be covered by the same object. Sometimes a graph representation is difficult to produce, as different sized objects impose different adjacency relationship between the same pair of vertices. For instances where such a representation can be formed, the covering problem reduces to a minimum clique cover problem as the piercing problem does.

In this thesis we consider the problem in the context of covering points with axis-parallel rectangles. If the covering rectangles are all of same description R, then the problem can be formulated as follows : For each point $p \in P$, put a rectangle of type R with p at its top-left corner. Thus we have a set of (possibly intersecting) rectangles of same size R. Now we need to find the minimum clique cover C of the corresponding rectangle intersection graph. For each clique $X \in C$, we put a rectangle with its bottom right corner at the common region of intersection of the members in X. Finding the minimum clique cover in a rectangle intersection graph of equal-sized rectangles is NP-Hard and 2-factor approximation is easy to set.

1.5 Maximum Independent Set of Rectangles

The Maximum Independent Set of Rectangles (MISR) problem can be stated as follows: given a collection $\mathcal{R}$ of n axis- parallel rectangles, find a maximum-cardinality subset of disjoint rectangles. The heavy rectangles in the following figure represents the maximum independent set of the rectangles. MISR is a special case of the classical Maximum Independent Set Problem, where the input is restricted to intersection graphs of axis-parallel rectangles. Due to its many applications, ranging from map labeling to data mining, (MISR) has received a significant amount of attention from various research communities. Since the problem is NP- hard, the main focus has been on the design of approximation algorithms.

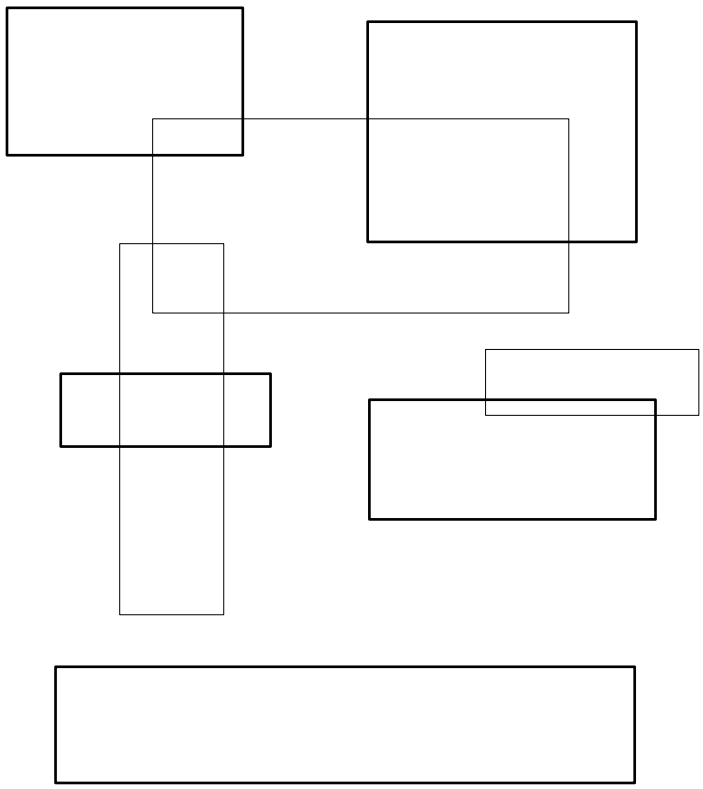

Figure 1.4: A demonstration of MISR

1.6 Maximum Clique of Rectangles

Though the minimum clique cover of rectangle intersection graph is NP-Hard, finding the largest clique can be solved in polynomial time.

A clique in a set of rectangles $\mathcal{R}$ is a subset of $\mathcal{R}$ in which for any 2 rectangles there exists an intersection between them. A clique C is said to be a maximal clique if it is not a subset of any other clique. We can state the Maximum Clique of Rectangles problem as : given a collection of $\mathcal{R}$ of n axis-parallel rectangles, find the maximum-cardinality subset $\mathcal{C}$ of $\mathcal{R}$ such that for any 2 rectangles in $\mathcal{C}$ there exists an intersection between them. In the following figure, the black square dot represents the area where maximum clique has formed. The other circular dots represents the maximal cliques.There can be at most $O(n^2)$ number of cliques in a rectangle intersection graph, where n is the number of rectangles.

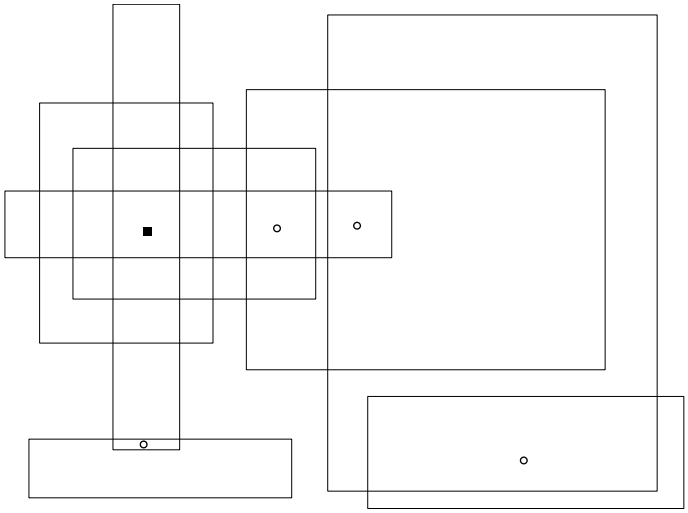

Figure 1.5: Maximum Clique Illustration

1.7 The Piercing or Stabbing Problem

A Piercing Set for n given objects in d-dimensional space $\mathbb{R}^d$ is a set of points such that each object contains at least one of the points in the set. The minimum cardinality of a piercing set is known as the Piercing Number. The general Piercing Problem asks for finding the piercing number of n given objects in $\mathbb{R}^d$. The piercing problem is sometimes termed as the Stabbing Problem as well. However stabbing problems may also consider the given objects by some specific geometric objects for example, lines. In our discussion, we shall restrict ourself to the axis-parallel rectangles of arbitrary sizes only.

The combinatorial counterpart of the piercing problem is the Hitting Set problem. The hitting set problem can be stated as : Let $\mathcal{C}$ be a collection of subsets of a set $\mathcal{U}$, find a subset $\mathcal{X} \subseteq \mathcal{U}$, with minimum cardinality such that for any subset $\mathcal{Y}$ in $\mathcal{C}$, $\mathcal{Y} \cap \mathcal{X} \neq \emptyset$. The *piercing problem* can be cast in a graph-theoretic setting as well. An *intersection graph* of the given objects is formed

by mapping each object to a different vertex and placing an edge between two vertices if the corresponding objects intersect, i.e., have at least one point in common. The piercing problem thus mapped to finding the minimum geometric cover of that intersection graph.

A Clique in a graph is a subgraph in which every pair of vertices are adjacent (i.e., connected by an edge) and a Clique Cover of a graph is a set of cliques such that each vertex is contained in at least one of the cliques. The *piercing problem* thus asks for a clique cover of the minimum size in the intersection graph.

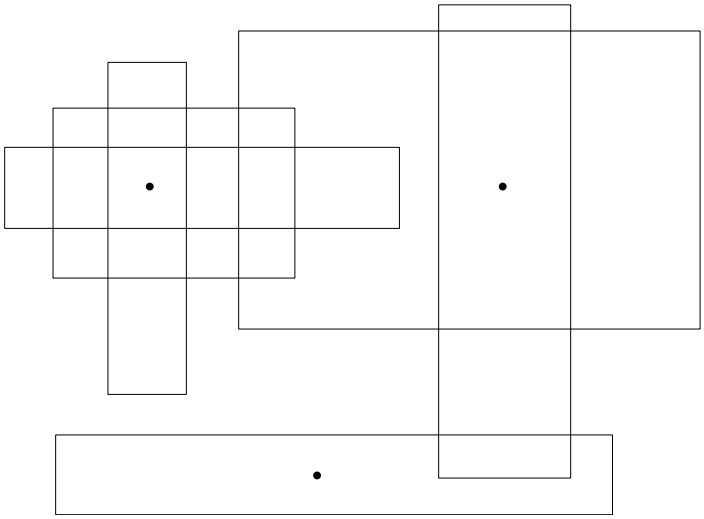

Figure 1.6: An example of the Piercing Set

1.8 Motivation for the Piercing Problem

The piercing problem has applications in facility location. In typical facility location problems, a collection of demand points, a parameter p and a distance function are given. The objective is to find a set of p supply points or facilities so that the maximum distance between a demand point and its nearest facility is minimized. This formulation is widely known as the p-center problem. The piercing problem addresses a different formulation of the facility location problem: given a set of demand regions and probable locations for facilities, the objective is to minimize the number of facilities to be established so that all demands are served. There is also another form of piercing problem known as the p-piercing problem. The number of points, p, to be used for piercing is given in advance, and the p-piercing problem asks to decide whether the set of given objects can be pierced using p points. However, the values of p for which solutions exist are usually very small.

1.9 Interval Piercing

The interval piercing problem is the one-dimensional case of the rectangle piercing problem. The well-known algorithm for finding the piercing number of a set of given intervals is quite simple and works in a greedy manner. The intervals are sorted by their right endpoints for convenience. The algorithm then chooses the leftmost right endpoint as the first piercing point and removes the intervals pierced by the point. The process of finding the leftmost right endpoint among the remaining intervals and removal of intervals pierced by it is repeated until no interval is left for piercing. The number of piercing points selected in this manner is optimum and is reported as the piercing number. The process of identifying the leftmost right endpoint and removal of intervals can be done in a single scan over the sorted list of intervals. For n intervals the scan takes only $O(n)$ time. The overall time is $O(n \log n)$ due to time required for presorting the intervals according to right endpoints. The algorithm can proceed in a right to left direction as well with the intervals sorted according to the left endpoints and the rightmost left endpoints chosen iteratively as piercing points. In the following figure, the vertical dashed lines represents the positions at which the intervals are optimally pierced.

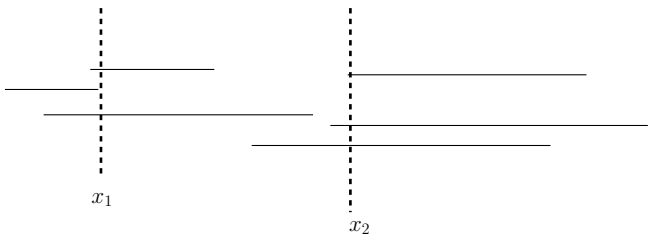

Figure 1.7: An example of Interval Piercing

1.10 The Complexity of Piercing

The covering problem was the first to receive attention. Tanimoto and Fowler investigated the problem of 2-dimensional covering with squares in the context of image processing: find the minimum number of square *patches* for storing information such that all points with information are contained in at least one of the patches. However, they advocated the use of heuristics, because even for the very special case of axis-aligned squares, the decision version of the covering (and packing) problem has been shown to be NP-complete by Fowler et al.. The proof is by reduction from the well known 3-SAT problem. The optimization versions of the problems are consequently NP-hard. The complexity result of the covering problem for axis-aligned squares easily carries over to piercing axis-aligned squares in the plane by a simple transformation: consider the points in the covering problem as the center of the axis-aligned squares in the piercing problem and replace the covering squares with their center points. The center points become a piercing set for the newly formed squares. Thus the piercing problem for axis-aligned identical squares in the plane is also NP-hard. More generally, the piercing problem for axis-aligned hyperrectangles in $\mathbb{R}^d$ is NP-

hard whenever $d \geq 2$. The one dimensional case involving intervals can be solved optimally in polynomial time as shown previously.

1.11 Related Works

The axis-parallel rectangles is always a point of great interest in the field of algorithms. For the Maximum Independent Set of Rectangles (MISR) problem, *Parinya Chalermsook* and *Julia Chuzhoy* gave $O(\log \log n)$ approximation algorithm in their paper *Maximum Independent Set of Rectangles*. They also considered a generalization of MISR to higher dimensions, where rectangles are replaced by d-dimensional hyper-rectangles. *Hiroshi Imai* and *Takao Asano* in their paper *Finding the Connected Components and a Maximum Clique of an Intersection Graph of Rectangles in the Plane* presented two problems on intersection graphs of rectangles in the plane. One is an $O(n \log n)$ algorithm for finding the connected components of an intersection graph of n rectangles. This algorithm is optimal to within a constant factor. The other is an $O(n \log n)$ algorithm for finding a maximum clique of such a graph. They also showed that the *k-colorability* problem on intersection graphs of rectangles is NP-complete. In the paper *Fast stabbing of boxes in high dimensions* by *Frank Nielsen* we can find a simple and efficient algorithm for stabbing a set $\mathcal{I}$ of n axis-parallel boxes in d-dimensional space with $c(\mathcal{I})$ points in output-sensitive time $O(dn \log c(\mathcal{I}))$ and linear space.

Chapter 2

Our Approach

2.1 Maximum Clique Algorithm using Horizontal Line Sweep

The method of Line Sweep is very widely used in the field of Computational Geometry. In computational geometry, a sweep line algorithm or plane sweep algorithm is a type of algorithm that uses a conceptual sweep line or sweep surface to solve various problems in Euclidean space. It is one of the key techniques in computational geometry. The idea behind algorithms of this type is to imagine that a line (a vertical or horizontal line) is swept or moved across the plane, stopping at some points. Geometric operations are restricted to geometric objects that either intersect or are in the immediate vicinity of the sweep line whenever it stops, and the complete solution is available once the line has passed over all objects.

The key idea is to scan a horizontal sweep-line across the axis-parallel rectangles from the topmost rectangle to the lowest one. Successive positions of the sweep-line are determined by the y-coordinates of the top and bottom sides of the rectangles sorted in descending order according to the y-coordinate which we may call as *events*. Before presenting the algorithm formally, we describe a brief outline of it. Initially, we have one empty list. At every step of the algorithm we maintain a list (Sweep Line Status list) of current active rectangles by storing its horizontal interval in a list sorted according to x-coordinate. Also, in the list we maintain a integer count that says the number of rectangles overlap on that interval at the current position of the sweep line.

We shall define a rectangle to be *inactive* if the current sweep line does not intersect it otherwise we shall say the rectangle to be *active*.

Whenever we meet a top boundary, we mark this rectangle as active. Let $\mathcal{I} = [\alpha, \beta]$ be the corresponding interval. We identify the set of intervals $I_1 ... I_m$ in the list with which $\mathcal{I}$ overlaps. Note that I_1 contains α and I_m contains β. $I_1 = [\phi_1, \psi_1]$ is split into two parts $I_1' = [\phi_1, \alpha]$, $I_1'' = [\alpha, \psi_1]$ and $I_m = [\phi_2, \psi_2]$ is split into two parts $I_m' = [\phi_2, \beta]$, $I_m'' = [\beta, \psi_1]$. The count field of I_1' will remain same as that of I_1 , I_1'' will be one more than that of I_1', I_m'' will remain

same as that of I_m and I'_m will be one more than that of I'_m. For all intervals $I_2, ..., I_{m-1}$, the count will be increased by 1.

There are many intricate and special cases where we need to handle them specially. Details of these cases shall be discussed later in this report.

Now we come to the case when we meet a bottom boundary of a rectangle. Firstly, we mark the rectangle to be inactive. Now if we do a careful observation we find that we must have one interval whose left x-coordinate is equal to the left x-coordinate of the current rectangle. Let the interval be $[x_i, x_j]$. Since the rectangle is leaving from the set of current active rectangles, we merge the interval with the left interval. Similarly, we perform such work while handling the right boundary. As we have pointed before, we postpone the discussions of the intricate and special cases until we present the formal algorithm. We also adjust the count of the intervals accordingly. Next we take a point just above the lower boundary and find the set of active rectangles that contain the point. These set of rectangles forms a maximal clique and we store its size. Finally we report the largest clique.

From now on, by rectangle incident on an interval, we mean that the horizontal interval of the rectangle and the interval have a common intersection interval.

2.1.1 The Maximum Clique Algorithm

As we have pointed before, the underlying idea is the basic line sweep methodology. Since in due course of line sweep we need to maintain the current list of x-intervals of the active rectangles, Doubly Linked List can be of great help for two reasons. Firstly, it is a dynamic data structure where we can add or delete intervals during the line sweep. Secondly, we can do the update operations efficiently. Other data structures may also work well but3 for our experimental simplicity we have opted for this data structure. The algorithm works in polynomial time.

In each node of the doubly linked list we maintain seven fields to denote the current status of the interval corresponding to the node.

1. Left : Stores the left boundary of the interval
2. Right : Stores the right boundary of the interval
3. Prev : Points to the previous interval in the list
4. Next : Points to the next interval in the list
5. Count : Stores the number of rectangles incident on this interval
6. RecEndCountL : Stores the number of rectangle (left/right) boundary that intersected the left end of this interval
7. RecEndCountR : Stores the number of rectangle (left/right) boundary that intersected the right end of this interval

When we encounter a top boundary during line sweep, there can arise twelve different cases as shown in the following figures. The upper line is an interval from the list of intervals we maintain during sweep. The rectangle below it is the rectangle whose status is going to change from *inactive* to *active*. When we encounter the lower boundary, we merge or adjust the node fields accordingly. For each interval in the list, we check each of the following cases and act accordingly. Algorithm 1 presents pseudocode of finding maximum clique in the rectangle intersection graph. Here, $N(i, j)$ denotes the node that corresponding to the interval $[i, j]$. At every instant the list is maintained as sorted according to x-coordinate of the intervals i.e if $N(i, j)$ and $N(k, l)$ be two nodes and $j <= k$, then $N(i, j)$ must be present before $N(k, l)$ in the list. When we mention insert a node in the list, the node has to be inserted at the appropriate position in the list and its fields are to be adjusted accordingly. The regular adjustments of the fields of a node are not shown to make the pseudocode more friendly towards basic conception rather than focusing at actual implementation.

2.1.2 The 12 cases in processing a top boundary

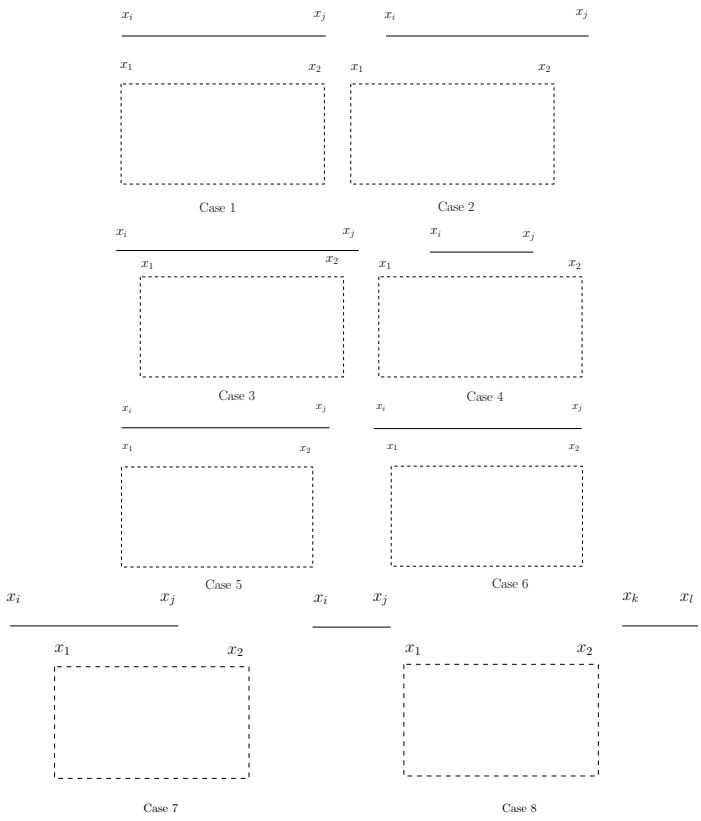

Figure 2.1: Maximal Clique Analysis : Case 1-8

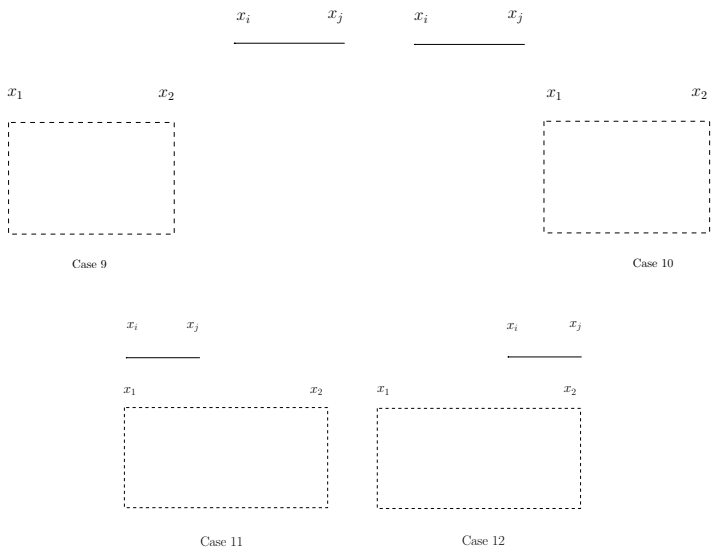

Figure 2.2: Maximal Clique Analysis : Case 9-12

Algorithm 1 Maximum Clique Algorithm for Axis-Parallel Rectangles

procedure MAXCLIQUE($\mathscr{R}$)

sort the horizontal boundaries, both upper and lower according to y-coordinate and store it in an array H;

create one empty list;
$MaxClique \leftarrow \phi$; $SizeMaxClique \leftarrow 0$;

for each boundary $(x_1, x_2) \in H$ **do**
if (x_1, x_2) an upper boundary of the rectangle r **then**

Change the status of the rectangle r from *inactive* to *active*;

if list is empty **then**
insert $N(x_1, x_2)$ in the list;
continue;
end if

for each interval node in the list **do**

Case 1 if $(x_i = x_1) \wedge (x_j = x_2)$ holds true
increment the count of the $N(x_i, x_j)$;

Case 2 if $(x_1 < x_i) \wedge (x_i < x_2 < x_j)$ holds true
if $N(x_i, x_j).Prev$ is empty or $N(x_i, x_j).Prev.Right < x_1$
insert a newnode $N(x_1, x_i)$ in the list;

Case 3 if $(x_i < x_1 < x_2 < x_j)$ holds true split the node
$N(x_i, x_j)$ into three nodes - $N(x_i, x_1)$, $N(x_1, x_2)$ and $N(x_2, x_j)$;

Case 4 if $(x_1 < x_i < x_j < x_2)$ holds true split the node
increment the count of the $N(x_i, x_j)$. If $N(x_i, x_j).Prev$
is empty then insert $N(x_i, x_1)$ else adjust fields of
$N(x_i, x_j).Prev$ accordingly. If $N(x_i, x_j).Next$ is empty
insert $N(x_2, x_j)$ else $N(x_i, x_j).Next$ accordingly;

Case 5 if $(x_i = x_1 < x_2 < x_j)$ holds true
split $N(x_i, x_j)$ into $N(x_i, x_2)$ and $N(x_2, x_j)$;

Case 6 if $(x_i < x_1 < x_2 = x_j)$ holds true
split $N(x_i, x_j)$ into $N(x_i, x_1)$ and $N(x_1, x_j)$;

Algorithm 2 Maximum Clique Algorithm for Axis-Parallel Rectangles Part 2

Case 7 if $(x_i < x_1 < x_j < x_2)$ holds true
split $N(x_i, x_j)$ into $N(x_i, x_1)$ and $N(x_1, x_j)$.
If $N(x_i, x_j).Next$ is empty insert $N(x_j, x_2)$;

Case 8 if the interval x_1, x_2 comes in between two
intervals $N(x_i, x_j)$ and $N(x_k, x_l)$, then insert
a new node $N(x_1, x_2)$ in the list;

Case 9 if $N(x_i, x_j).Prev$ is empty and $x_2 < N(x_i, x_j).Left$
insert a new node $N(x_1, x_2)$ in the list;

Case 10 if $N(x_i, x_j).Next$ is empty and $N(x_i, x_j).Right < x_1$
insert a new node $N(x_1, x_2)$ in the list;

Case 11 if $(x_i = x_1 < x_j < x_2)$ holds true
increment $N(x_i, x_j).Count$ and insert $N(x_i, x_2)$ if
$N(x_i, x_j).Next$ ie empty;

Case 12 if $(x_1 < x_2 < x_j = x_2)$ holds true
increment $N(x_i, x_j).Count$ and insert $N(x_1, x_i)$ if
$N(x_i, x_j).Prev$ ie empty;

end for

else if (x_1, x_2) is lower boundary **then**

take a point just above the boundary and find the set of active rectangles Y that contain the point;

if $|Y| > SizeMaxClique$, then $SizeMaxClique \leftarrow |Y|$ and store the clique in $MaxClique$;

decrement the $Count$ of the nodes with which (x_1, x_2) overlaps;

merge two intervals $N(x_i, x_j)$ and $N(x_j, x_k)$ if
$N(x_i, x_j).RecEndCountR == N(x_j, x_k).RecEndCountL == 0$;

if $Count$ of any node becomes zero, delete the node from the list;

end if

end for

return $MaxClique$;

end procedure

2.2 The Greedy Clique Cover Algorithm

In this section we present a simple approach to obtain the clique cover of a set of axis-parallel rectangles. This algorithm calls the maximum clique algorithm as discussed in previous section. Let $ADJ(C)$ of a maximal clique C denotes the set of all rectangles R_i that forms the clique C. Now the algorithm goes as follows.

Algorithm 3 Greedy Clique Cover Algorithm for Axis-Parallel Rectangles

procedure GREEDYCLIQUECOVER

1. Initially GCC is empty and R contains all rectangles;
2. Use Algorithm 1 to compute a clique C of maximum cardinality;
3. Stab all the rectangles in C by a pin p;
4. Set $R = R \backslash ADJ(C)$ and $GCC = GCC \cup p$;
5. Repeat Step 2 and 3 recursively until R is empty;
6. Return GCC which is the stabbing set of R;

end procedure

2.3 Finding maximum Independent Set of the Rectangles

Here, we have implemented the brute force approach described by Wagner et al. The approach is based on finding simplicial-rectangles. Let us consider an ordering of rectangles $R_1, ..., R_n$. A rectangle R_i is said to be simplicial if all the neighbors of R_i denoted by $N(R_i)$, $R_1..., R_{i-1}$ forms a clique. Two rectangles will be said to be neighbors of each other if they intersects. Let $N(R_i)$ denotes the set of all neighbors of a rectangle R_i. The following algorithm returns the Maximum Independent Set (MIS) of our input axis-parallel rectangles.

Algorithm 4 MIS Algorithm for Axis-Parallel Rectangles

procedure MISR(R)

1. Initially MIS is empty and R contains all rectangles;

2. If there is any simplicial-rectangle $R_i \in R$ then set $MIS = MIS \cup R_i$ and $R = R \backslash (R_i \cup N(R_i))$, else delete $R_j \in R$ with $max(|N(R_j)|)$ (The part in else gets executed iff there is no simplicial-rectangle and $R \neq \phi$).

3. Repeat Step 2 recursively until R is empty.

4. return MIS

end procedure

2.4 Divide and Conquer Algorithm for Interval Stabbing

The methodology consists in applying the traditional DIVIDE-AND-CONQUER strategy to the set of intervals $\mathscr{I} = \{[a_1, b_1], ..., [a_n, b_n]\}$. However, it differs from the traditional DIVIDE-AND-CONQUER by filtering the recursive subsets. The algorithm mentioned below returns an optimal stabbing point set $\mathscr{P}$ of $\mathscr{I}$. The algorithm is *output-sensitive* in nature. The running time complexity is of $O(n \log c)$ where c is the size of the stabbing set. Also the algorithm uses linear space.

Algorithm 5 Divide-and-Conquer for Interval Stabbing

procedure DANDCI($\mathscr{I}$)
  if $n = 1$ **then**
    return $\mathscr{P} = \{a_1\}$;
  else
    if $\{a_1 = a_2 = ... = a_n\}$ **then return** a_1;
    compute median m of $\{a_1, ..., a_2\}$;
    $\mathscr{I}_r \leftarrow \{[a_i, b_i] | a_i \geq m\}$;
    $\mathscr{I}_l \leftarrow \mathscr{I} \setminus \mathscr{I}_r$;
    $\mathscr{P}_r \leftarrow$ DANDCI($\mathscr{I}_r$);
    $q \leftarrow \min \mathscr{P}_r$ be the leftmost stabbing point of $\mathscr{I}_r$;
    $\mathscr{I}'_l \leftarrow \{[a_i, b_i] | b_i < q\}$;
    $\mathscr{P}_l \leftarrow$ DANDCI($\mathscr{I}'_l$);
    return $\mathscr{P} \leftarrow \mathscr{P}_r \cup \mathscr{P}_l$;
  end if
end procedure

2.5 Divide and Conquer Algorithm for Rectangle Stabbing

For axis-aligned boxes or hyperrectangles in d-dimensional space, Nielsen provides an algorithm for piercing based on a simple divide-and-conquer approach. The algorithm computes a *median* axis-aligned hyperplane and partitions the set of input hyperrectangles into three sets — one set consisting of the hyperrectangles intersected by the hyperplane, the second one consisting of hyperrectangles lying entirely in one halfspace determined by the hyperplane and the remaining set con- sisting of hyperrectangles lying in the other halfspace. The piercing problem for the first set reduces to a piercing problem involving $(d-1)$-dimensional hyperrectangles defined by the median hyperplane. Recursive solutions are computed for the latter two sets. The base case is the interval stabbing problem and is solved optimally. The running time of the algorithm is $O(n \log^{d-1} n)$ In this section, we have modified the DIVIDE-AND-CONQUER algorithm by Nielsen for stabbing n-dimensional boxes to work for axis-parallel rectangles.which is much better than the ϵ-approximation schemes. However, the approximation factor achieved is $O(\log^{d-1} n)$.

Algorithm 6 Divide-and-Conquer for Rectangle Stabbing

procedure StabRectangles($\mathscr{R}$)
  $\mathscr{I}^{(x)} \leftarrow$ set of x-coordinate intervals of all rectangles;
  $\mathscr{P}_x(\mathscr{I}) \leftarrow$ DandCI($\mathscr{I}^{(x)}$) ;
  Sort $\mathscr{P}_x(\mathscr{I})$;
  procedure DandCR($\mathscr{R}$)
    select the value m of the median stabbing point of x-intervals of $\mathscr{R}$ using $\mathscr{P}_x(\mathscr{I})$;
    $H_m \leftarrow (x = m)$;
    $\mathscr{R}_1 \leftarrow$ rectangles that do not cross H_m and are to the left of H_m;
    $\mathscr{R}_2 \leftarrow$ rectangles that do not cross H_m and are to the right of H_m;
    $\mathscr{R}_m \leftarrow$ rectangles intersecting H_m;
    if $|\mathscr{P}_x(\mathscr{R}_1)| >= 1$ **then call** $\mathscr{P}_l \leftarrow$ DandCR($\mathscr{R}_1$);
    if $|\mathscr{P}_x(\mathscr{R}_2)| >= 1$ **then call** $\mathscr{P}_r \leftarrow$ DandCR($\mathscr{R}_2$);
    Stab the rectangles of $\mathscr{R}_m$ by piercing the y-coordinate intervals $\mathscr{R}'_m$, **call** $\mathscr{P}_m \leftarrow$ DandCI($\mathscr{R}'_m$);
    return $\mathscr{P} \leftarrow \mathscr{P}_l \cup \mathscr{P}_r \cup \mathscr{P}_m$;
  end procedure
end procedure

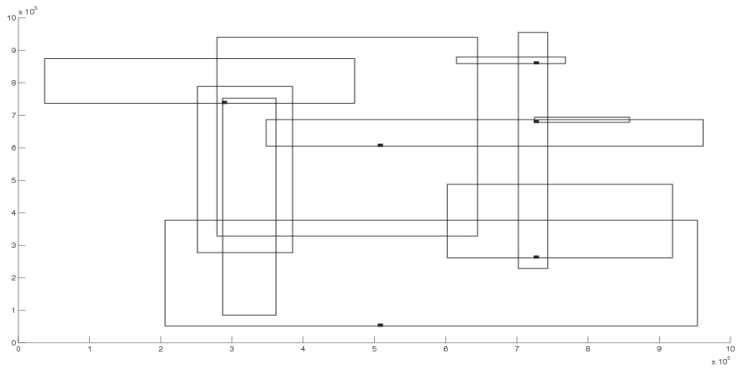

Figure 2.3: Our implementation for the Divide and Conquer Rectangle Stabbing

Chapter 3

Experimental Results

3.1 Experimental Steps Carried Out

1. We have generated a set of n random rectangles in a 1000000 x 1000000 grid for different values of n (5,50,75,100,200,300,400,500,600,700,800,900, 1000). For each value in the set of n and we have computed the following parameters under study.
   (a) Maximum Clique
   (b) Greedy Clique Cover (GCC)
   (c) Maximum Independent Set (MIS)
   (d) Clique Cover computed using Divide and Conquer strategy (DCC)
2. We have reported average of 20 results for each parameter.
3. Finally we have plotted the ratio $|GCC|/|MIS|$ for different values of n.

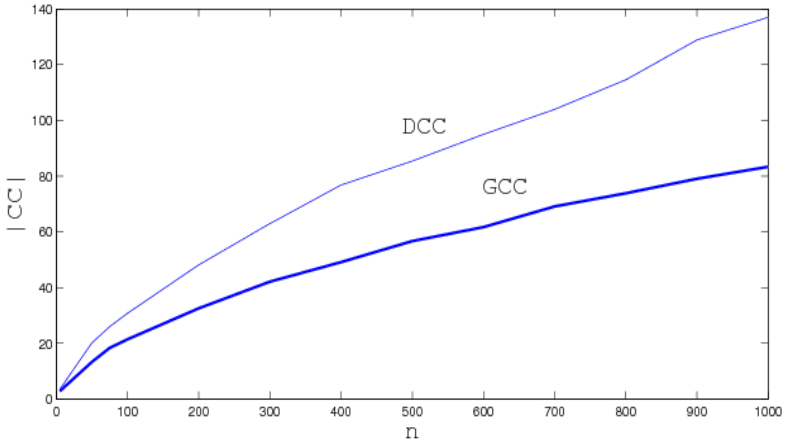

Figure 3.1: The Comparison between the approaches.

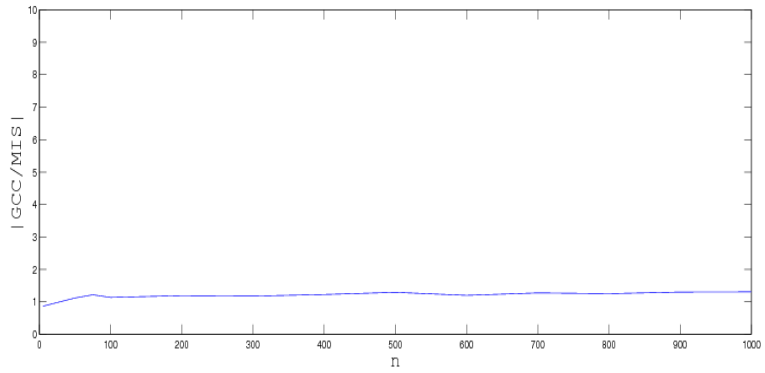

Figure 3.2: Plot of $|GCC|/|MIS|$

n	**|GCC|**	**|MIS|**	**R = |GCC|/|MIS|**
5	2.8500	3.3000	0.8636
50	13.3000	11.9000	1.1176
75	18.3500	15.1000	1.2152
100	21.4500	18.9000	1.1349
200	32.5500	27.4000	1.1880
300	42.1000	35.8500	1.1743
400	49.1500	40.1500	1.2242
500	56.7000	43.9500	1.2901
600	61.7000	51.5000	1.1981
700	69.1000	54.2500	1.2737
800	73.8500	59.1000	1.2496
900	79.0500	60.9000	1.2980
1000	83.35	63.8500	1.3054

Figure 3.3: Performance measurement of our Greedy Approach

n	|DCC|
5	3.5000
50	20.1500
75	26.0000
100	30.8000
200	48.1500
300	62.9500
400	76.8000
500	85.4000
600	94.9500
700	103.9500
800	114.5500
900	128.8500
1000	137.0000

Figure 3.4: Performance measurement of our Divide-And-Conquer Approach

3.2 Conclusion

Figure 3.1 shows the performance of GCC and DCC for different values of n. Finally we have observed that the ratio of the size of the clique cover and maximum independent set for a set of n randomly generated rectangles is less than 2 for reasonable values of n.

Plot 3.2 suggests that $\frac{GCC}{MIS} \leq 2$ for almost all values of n when the rectangles are generated randomly. Also, $\frac{MIS^*}{CC^*} \leq 1$. In other words, $MIS^* \leq CC^* \leq CC \leq 2 \times MIS \leq 2 \times MIS^*$. Thus, GCC is a 2 factor approximation of the MIS problem for randomly generated rectangles.

Again $CC^* \geq MIS^* \geq MIS \geq \frac{GCC}{2} \geq \frac{CC^*}{2}$. Thus, any solution for the MIS problem is a 2 factor approximation of the minimum clique cover problem for randomly generated rectangles.

3.3 Future Scope

The experiment can further be extended to geometrical objects which can be generated randomly. The same ratio may be observed in those type of geometric objects as we did in this dissertation.

Bibliography

[1] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.

[2] Parinya Chalermsook, Julia Chuzhoy, *Maximum Independent Set of Rectangles*

[3] P. K. Agarwal, M. van Kreveld and S. Suri, *Label placement by maximum independent set in rectangles*, Computational Geometry Theory and Applications, vol.11, pp. 209218, 1998.

[4] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars, *Computational Geometry (3rd revised ed.)*, Springer-Verlag

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, second edition, 2001.

[6] Boris Aronov, Esther Ezra, and Micha Sharir, *Small-size epsilon-nets for axis-parallel rectangles and boxes*, STOC, 2009, pp. 639–648.

[7] V. Chvȧtal, *A greedy heuristic for the set-covering problem*, SIAM Journal on Computing **12** (1983), 759–776.

[8] Hiroshi Imai and Takao Asano, *Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane*, J. Algorithms **4** (1983), no. 4, 310–323.

[9] Frank Nielsen, *Fast stabbing of boxes in high dimensions*, Theor. Comput. Sci. **246** (2000), no. 1-2, 53–72.

[10] Frank Wagner, Alexander Wolff, Vikas Kapoor, and Tycho Strijk, *Three rules suffice for good label placement*, Algorithmica **30** (2001), no. 2, 334–349.