Approximation Algorithms for Rectangle Piercing Problems

by

Abdullah-Al Mahmood

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2005

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Piercing problems arise often in facility location, which is a well-studied area of computational geometry. The general form of the piercing problem discussed in this dissertation asks for the minimum number of facilities for a set of given rectangular demand regions such that each region has at least one facility located within it. It has been shown that even if all regions are uniform sized squares, the problem is NP-hard. Therefore we concentrate on approximation algorithms for the problem. As the known approximation ratio for arbitrarily sized rectangles is poor, we restrict our effort to designing approximation algorithms for unit-height rectangles. Our ϵ-approximation scheme requires $n^{O(1/\epsilon^2)}$ time. We also consider the problem with restrictions like bounding the depth of a point and the width of the rectangles. The approximation schemes for these two cases take $n^{O(1/\epsilon)}$ time. We also show how to maintain a factor 2 approximation of the piercing set in $O(\log n)$ amortized time in an insertion-only scenario.

Acknowledgements

I am grateful to my thesis supervisor Timothy M. Chan for guiding me in my research. His insightful comments and suggestions have helped me immensely in writing this thesis.

I am also grateful to the readers of my thesis for their valuable opinions.

Contents

List of Figures

Chapter 1

Introduction

Computational geometry focuses on problems involving geometric objects ranging from simple geometric primitives like *points* and *intervals* to complex higher-dimensional *polyhedra*. While some geometric problems motivate investigation for a solution in a general setting, many *combinatorial problems* are studied in the geometric context as well. The latter is applicable to problems that are difficult to solve in a general setting but special properties of geometric objects render the geometric instance more *tractable*. Geometric instances or analogues are found to be interesting also because they can be related more closely to real problems than their combinatorial counterparts. Furthermore, geometric objects can be subjected to reasonable *restrictions*, *simplifications* and *transformations* while maintaining adherence to the original problem. This dissertation focuses on one such geometric problem – the *piercing* problem. The piercing problem is discussed by some researchers along with the *packing* problem [Cha03, KNM03], which in turn is related to the *covering* problem [FPT81, HM85]. The following sections provide a brief introduction to these related *geometric optimization* problems.

1.1 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 the n given objects in $\mathbb{R}^d$. The piercing problem is sometimes termed as the *stabbing* problem as well. However stabbing problems can refer to intersecting the given objects with geometric objects other than points, *e.g.*, lines [HM91, GIK02, KS04]. In this thesis, stabbing and piercing will refer to *hitting* objects with points only. An instance of 2-dimensional piercing problem for a set of rectangles with a minimum piercing set is shown in figure 1.1.

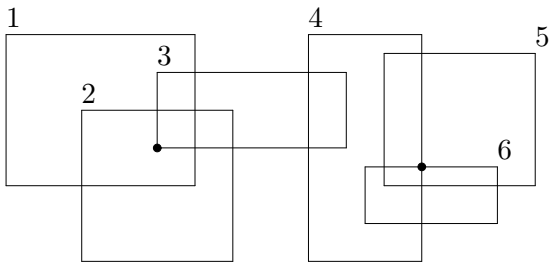


Figure 1.1: Rectangle piercing in 2 dimensions

The combinatorial counterpart of the piercing problem is the *hitting set* problem [GJ79]: given a collection C of subsets of a finite set S, find a subset $X \subseteq S$ with minimum cardinality such that for any subset S_i in C, $S_i \cap X \neq \emptyset$. 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 (see figure 1.2). A *clique* in a graph is a subgraph in which every pair of vertices are *adjacent* (*i.e.*, connected by an edge)

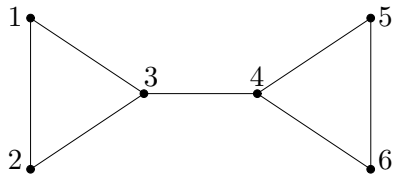


Figure 1.2: Intersection graph for set of rectangles in figure 1.1

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 intersection graph using minimum number of cliques. For arbitrary graphs the problem is widely known as *minimum clique cover*.

1.2 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. Although prohibiting translation and rotation of the objects make the nomenclature somewhat misleading, the term is in wide use in literature [FPT81, HM85, Cha03]. The number of objects in the largest sub-collection is called the *packing number*. In general the piercing number (see section 1.1) 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 placeholders 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 [FW91]. This is a special case of the *planar geometric packing* problem involving rectangles. The general planar geometric packing problem allows arbitrarily shaped objects in 2-dimensional space $\mathbb{R}^2$. Figure 1.3 shows an instance of planar geometric packing problem involving arbitrarily sized rectangles. The four rectangles with darker boundary can be *packed* together.

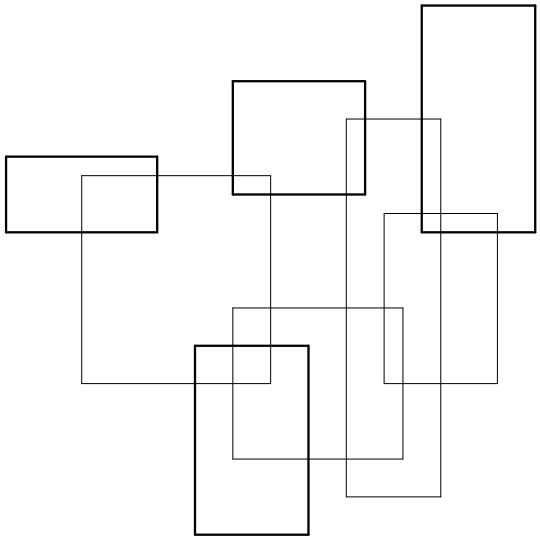


Figure 1.3: Planar rectangle packing

The combinatorial analogue for geometric packing is the *set packing* problem [GJ79] 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 (see section 1.1). The maximum independent set problem asks for a subset, of maximum cardinality, of vertices such that no two vertices in the set are adjacent.

1.3 Geometric Covering

For a collection of n given points in d-dimensional space $\mathbb{R}^d$, the general *geometric covering* problem asks for a set of geometric objects 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 particular, special cases of the covering problem are equivalent to corresponding instances of the piercing problem. 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 [HM87]. Figure 1.4 shows an instance of planar geometric covering with squares. The dashed lines indicate square boundaries used for the cover.

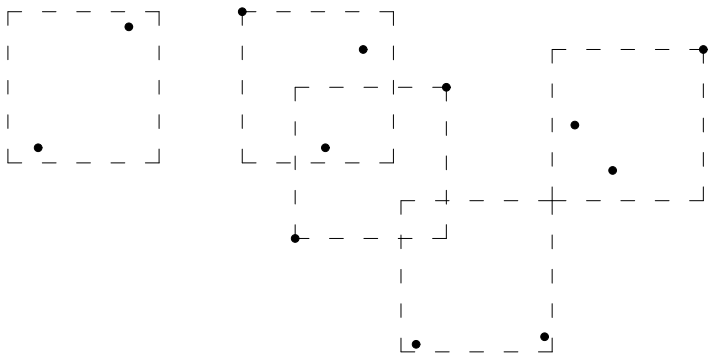


Figure 1.4: Covering with squares in 2 dimensions

Geometric covering is analogous to the well known *set cover* problem [GJ79, CLRS01]: 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 (see section 1.1). 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 (see section 1.1) as the piercing problem does.

1.4 Interval Piercing

The interval piercing problem is the one-dimensional case of the general piercing problem. It is related to piercing in higher dimensions. 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 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 [CLRS01] 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 [CLRS01]. 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.

1.5 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 [AS98]. 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 [SW96]. However, the values of p for which solutions exist are usually very small.

1.6 Organization of the Thesis

We have introduced in this chapter the basic concepts and definitions used in the rest of the thesis. In chapter 2 we mainly discuss the complexity and earlier work related to the piercing problem. In chapter 3, which embodies a significant portion of our contribution, we discuss *approximation schemes* for the piercing problem in the case of a set of unit-height rectangles under different conditions. We consider the piercing problem in a dynamic context in chapter 4. In this chapter we discuss earlier work on interval piercing in one dimension, which is related to approximation algorithms for unit-height rectangles. We show how to efficiently maintain the

piercing set for intervals in an insertion-only case. Part of our contributions in chapters 3 and 4 will be presented at the Canadian Conference on Computational Geometry 2005 [CM05]. Finally we present some open questions in chapter 5.

Chapter 2

Background

2.1 The Complexity of Piercing

The covering problem was the first to receive attention [Tan79]. Tanimoto and Fowler investigated the problem of 2-dimensional covering with squares in the context of image processing [TF80]: 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.* [FPT80, FPT81]. The proof is by reduction from the well known 3-SAT problem. The optimization versions of the problems are consequently NP-hard (for a discussion on NP-completeness, refer to the text by Garey and Johnson [GJ79]). 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 (see section 1.4 for details).

2.2 Approximation Algorithms and Schemes

An algorithm with sub-exponential running time has not yet been found for any of the NP-complete problems. Research efforts for these problems are therefore shifted towards finding "near-optimal" solutions in polynomial time. Algorithms for finding near-optimal solutions are called approximation algorithms. *Approximation ratio* provides a measure of the quality of the solutions produced by these algorithms. For a problem with any input of size n and the cost or value of the optimal solution being Z, an approximation algorithm is said to have approximation ratio $\rho(n)$ if Z^*, the cost or value returned by the algorithm, satisfies

$$\max\left(\frac{Z^*}{Z}, \frac{Z}{Z^*}\right) \leq \rho(n),$$

i.e., Z^* is within a factor $\rho(n)$ of Z [CLRS01]. The solution returned by the algorithm is called a $\rho(n)$-approximation.

Definition 2.1 (ϵ-approximation algorithm) *We define an approximation algorithm to be an ϵ-approximation algorithm if the approximation ratio of the algorithm is $1 + \epsilon$.*

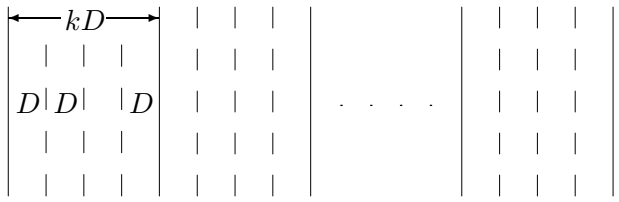
Definition 2.2 (ϵ-approximation scheme) *We define an approximation algorithm to be an ϵ-approximation scheme if the algorithm takes $\epsilon > 0$ as part of its input, and the solution returned by the algorithm for a fixed value of ϵ is a $(1+\epsilon)$-approximation.*

The running time of an ϵ-approximation scheme increases with $\frac{1}{\epsilon}$ as well as the input size. If the nature of growth of the running time is polynomial both in the size of the input and $\frac{1}{\epsilon}$, then the scheme is called a *fully polynomial time approximation scheme* (FPTAS). If the running time is polynomial only in input size then the scheme is called a *polynomial time approximation scheme* (PTAS). The problem of covering with squares and consequently the piercing problem for squares are NP-complete in the strong sense, and there are no FPTAS for these problems, nor for packing squares in the plane [GJ79, HM85]. The aim of further research is therefore the design of PTAS for these problems and their extensions.

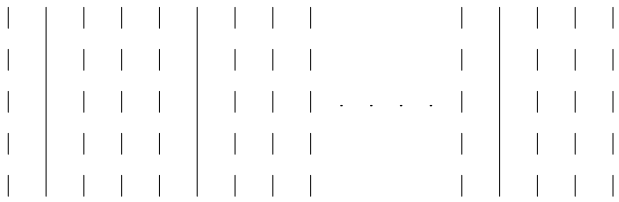
2.2.1 The Shifting Technique

Hochbaum and Maass provide a generic technique of grid shifting that can be applied to a broad range of problems in covering, packing and piercing [HM85]. They consider the square packing problem, disk covering problem and covering with squares problem arising respectively in VLSI layout design, facility location (see section 1.5) and image processing (see section 2.1). They demonstrate the technique for covering points with disks of diameter D. The whole extent of the given points is divided into semi-open vertical strips of width D. The strips are closed on the left and open on the right. A shifting parameter k is selected and k consecutive strips are considered as a group. The grouping strategy produces a partition of the area into semi-open regions of width kD. Evidently there can be

k ways of partitioning the area into groups of k strips, where each partition can be transformed into another by shifting the partition boundary to a distance D. After k shifts in a single direction, *e.g.*, to the right only, each partition is repeated.



(a) Initial partition



(b) Partition shifted to the right by D

Figure 2.1: The shifting technique

Thus a finite number of "scans" over the possible partitions allows selection of a solution *close* to the optimal. A local algorithm for each group is run and in each partition the solutions of all groups are combined together. The local algorithm can be either an approximation algorithm for one group of strips or an exact algorithm that solves the problem in that group. The final solution is obtained by choosing the "best" solution among all possible partitions.

The key observation for the disk cover problem (and the other two as well) is the near-decomposability of solutions within partition boundaries. Although a number of disks can cover points in two adjacent groups, no disk can straddle partition boundaries in two different shifted partitions. Combining the argument with the strategy of choosing the minimum number of disks among all shifted partitions, they are able to prove an approximation ratio for the technique.

Lemma 2.1 (Hochbaum and Maass' Shifting Lemma) *If ρ_{local} is the approximation ratio of a local algorithm and k is the shifting parameter then the shifting strategy provides an approximation ratio ρ within a factor of $(1 + \frac{1}{k})$ of ρ_{local}, i.e.,*

$$\rho \leq \rho_{local}\left(1 + \frac{1}{k}\right).$$

If the local algorithm is exact and $k = \frac{1}{\epsilon}$, the shifting technique yields an ϵ-approximation scheme. The idea of the shifting technique is very similar to the one introduced by Baker for approximation algorithm for some NP-complete problems involving planar graphs [Bak94]. Baker removes vertices based on *level* and partitions the graph. Solutions of partitions are combined using *dynamic programming* in order to construct final solution. The ϵ-approximation scheme of Hochbaum and Maass for covering with disks problem requires $n^{O(1/\epsilon^2)}$ time. The running time of ϵ-approximation schemes for covering with squares (or uniform sized boxes) and square packing is same.

Feder and Greene later consider the covering problem in the context of clustering [FG88] and Gonzalez considers the covering problem in the context of facility location [Gon91]. They consider shifting in vertical direction instead of horizontal, and both works make use of dynamic programming for solving the local covering problem within a slab, the height of which is determined by the shifting parameter. After application of the shifting algorithm the running time of the ϵ-approximation schemes turns out to be $n^{O(1/\epsilon)}$. In a relatively more recent work, Chan considers the rectangle packing problem, *i.e.*, maximum independent set problem in the rectangle intersection graphs and provides an $n^{O(1/\epsilon)}$ time ϵ-approximation scheme for rectangles with unit height [Cha04]. This scheme also uses dynamic programming for exact local solution and approximates the optimal solution using the shifting technique.

2.2.2 Other Divide-and-Conquer Algorithms

For axis-aligned boxes or *hyperrectangles* in d-dimensional space, Nielsen provides an algorithm for piercing based on a simple divide-and-conquer approach [Nie00]. 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 consisting 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 $O(n \log^{d-1} n)$, which is much better than the ϵ-approximation schemes. However, the approximation factor achieved is $\log_2^{d-1} n$.

A more advanced divide-and-conquer approach based on *geometric separators* has been proposed by Chan for *fat* objects [Cha03]. Chan provides the following definition for fat objects:

> A collection $\mathcal{C}$ of objects is fat if for any r and size-r box R, we can choose a constant number c of points such that every object that intersects R and has size at least r contains one of the chosen points.

This definition includes, but is not limited to, collections of boxes with bounded aspect ratio. The shifting technique is not suitable for the piercing problem involving fat objects of arbitrary size. Chan's algorithm computes an ϵ-approximation of the piercing number for a collection of fat objects in $n^{O(1/\epsilon^d)}$ time. An earlier algorithm for approximating the piercing set of fat objects is due to Efrat *et al.* [EKNS00].

Their algorithm produces a constant factor approximation and uses incremental elimination of objects from the collection.

Chapter 3

Piercing Unit-Height Rectangles

The piercing number for arbitrary rectangles is not known to be approximable within a constant factor in polynomial time. Hence we investigate the piercing problem for rectangles with some restrictions. The focus is on rectangles that are not restricted by uniformity of size in all dimensions as in the case considered by Hochbaum and Maass [HM85]. However, in order to accommodate "thin" and "long" rectangles, instead of *fat* objects of varying sizes [Cha03], we restrict one of the dimensions of all the rectangles, either height or length, to be same. For convenience, we shall discuss only the unit-height case, since the analysis for the other dimension, *i.e.*, length, will be similar, and a set of uniform height rectangles can be easily scaled to a set of unit-height rectangles.

3.1 PTAS with Depth Restriction

The advantage of all rectangles being of unit-height is that the problem may be subjected to divide-and-conquer strategies like the shifting technique (see section 2.2.1)

using horizontal dividing lines. In order to do so we, in principle, want to solve subproblems in a defined subregion efficiently, and dynamic programming is a powerful tool for this task. Noticeably, the domain of the problem instance is almost the same as that of the packing problem considered by Agarwal *et al.* [AvKS98] and later by Chan [Cha04]. However the dynamic programming paradigm does not readily yield a solution for our problem. The difficulty lies in finding a family of constant-sized sets of rectangles that can characterize a feasible solution to the problem. Such sets are necessary for formulating a recursive definition to be used in the dynamic programming solution. Taking this factor into account, we first restrict the problem to those instances where a point is not allowed to be contained by an arbitrary number of rectangles. We adopt the notion of *depth* (used also by Chan [Cha04]).

Definition 3.1 (Depth of a Point) *The depth of a point is the maximum number of rectangles containing the point.*

We propose a PTAS for the piercing problem for the special case of a set of unit-height rectangles where the maximum depth of a point is a constant. In terms of the intersection graph (see section 1.1) of the rectangle set, this condition restricts the largest clique size, *i.e.*, the maximum possible number of vertices in a clique of the graph. From an application point of view, this scenario is helpful if a large number of demand regions do not overlap and each of the supply points can serve up to a certain number of demand regions.

Lemma 3.1 *If a set of n axis-parallel rectangles can be stabbed by k horizontal lines where $k \geq 1$, then the piercing number of the rectangles can be computed in $O(n \log n + k\Delta n 2^{2k\Delta})$ time, where Δ is the maximum depth.*

Proof: Let $R = \{R_1, R_2, R_3, \ldots, R_n\}$ be a set of axis-parallel rectangles that can be stabbed by k horizontal lines. The piercing set of R can be found by dynamic programming. Let the set of x-coordinates of the corner points (and thereby of the intersection points) of the rectangles in R be $\{a_1, a_2, a_3, \ldots, a_m\}$. Without loss of generality, say $a_1 < a_2 < \cdots < a_m$.

At each vertical line $x = a_i$, we want to find the piercing number of the rectangles that have their left endpoint on or to the left of the line. The index of the current vertical line is an obvious parameter for the dynamic programming formulation. If we know the optimal solution for the previous vertical line, we can try to combine the solution with the optimal one for rectangles beginning on the current vertical line. However, the points on the current line may stab rectangles considered in the solution of the previous line. In figure 3.1, the rectangle drawn in dashed line is considered for piercing by the line on the left of $x = a_i$ and pierced by the new piercing point on $x = a_i$ as well. Therefore we need to add a second parameter

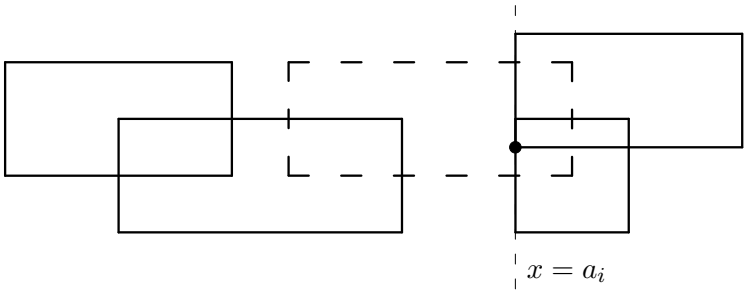


Figure 3.1: Piercing point causing rectangle exclusion

to the dynamic programming formulation for sets of rectangles that we want to exclude from stabbing.

We can create a dynamic programming table A for each of the vertical lines $x = a_i$ and each subset of rectangles that intersect any of these lines. Since the rectangles can be stabbed by k horizontal lines, the maximum cardinality of such a

set will be $k\Delta$. For the vertical line $x = a_i$ and a set S of rectangles, the dynamic table entry $A[i, S]$ indicates the minimum number of points that stab all rectangles with left endpoint on or to the left of the line $x = a_i$ except the rectangles in S. We let S_i denote the set of all rectangles intersected by $x = a_i$ and also let C_S be the minimum number of points required to stab S. The table A can be filled up as follows:

1. Base case: For a set $S \subseteq S_1$, set $A[1, S] \leftarrow C_{S_1 \backslash S}$.

2. For $i \leftarrow 2$ to m do

   For a set $S \subseteq S_i$, set

   $$A[i, S] \leftarrow \min_{S'}(A[i-1, S_{i-1} \cap (S \cup S')] + C_{S' \cup ((S_i \backslash S_{i-1}) \backslash S)})$$

   where the minimum is taken over all subsets $S' \subseteq S_i \backslash S$.

The cardinality of the minimum piercing set is $A[m, \emptyset]$.

While computing $A[i, S]$, all possible subsets S' of $S_i \backslash S$ are combined with the restricted set S for exclusion on the line $x = a_{i-1}$ (*i.e.*, to the left of the line $x = a_i$). By taking the intersection of $S \cup S'$ with S_{i-1}, we omit rectangles that have their left endpoint on the line $x = a_i$ from exclusion. The entry $A[i-1, S_{i-1} \cap (S \cup S')]$ thus provides the piercing number of the rectangles that have their left endpoint to the left of the line $x = a_i$ except any rectangle in $S \cup S'$. Now we must pierce the rectangles in S' separately. In addition, we must also pierce rectangles that have their left endpoint on the line $x = a_i$ except for rectangles in S (*i.e.*, $(S_i \backslash S_{i-1}) \backslash S$). The piercing number of $S' \cup ((S_i \backslash S_{i-1}) \backslash S)$ can be computed from the intervals formed by the rectangles on the line $x = a_i$ using the algorithm for interval piercing (see section 1.4). The final value of $A[i, S]$ is correct since all possible exclusions are considered and the minimum value is chosen.

Evidently there are $O(n)$ vertical lines and for each vertical line there are $O(2^{k\Delta})$ candidate subsets of rectangles. Each table entry can be filled up in $O(k\Delta 2^{k\Delta})$ time, since in the worst case there are $O(2^{k\Delta})$ sets to choose from for minimizing the entry and intersection can be performed in $O(k\Delta)$ time. Thus filling up the necessary entries of the table requires $O(k\Delta n 2^{2k\Delta})$ time. If the members of a subset S' are generated based on their vertical extent (or vice versa) then a $C_{S' \cup ((S_i \setminus S_{i-1}) \setminus S)}$ value can be computed (or pre-stored) in $O(k\Delta)$ time using the algorithm in section 1.4, since $|S| \leq k\Delta$. Thus enumeration of the sets and computation of their extent as well as C values can be done in $O(k\Delta n 2^{k\Delta})$ time. Including the $O(n \log n)$ time required for sorting, the overall running time of the algorithm is $O(n \log n + k\Delta n 2^{2k\Delta})$. The space requirement, mainly due to the storage for the dynamic programming table, is $O(n^2 2^{k\Delta})$. □

Theorem 3.1 *Fix an integer constant $k \geq 1$. For a set of n axis-parallel unit-height rectangles with constant maximum depth of points Δ, the cardinality of its minimum piercing set can be approximated within a factor of $\left(1 + \frac{1}{k}\right)$ in $O(n \log n + k^2 \Delta n 2^{2k\Delta})$ time.*

Proof: The approximation is based on the idea of shifting as outlined in the following algorithm:

1. For $i \leftarrow 0$ to $k - 1$ do

   Define a nonempty subset $R^{(i,j)}$ of the rectangles for each possible $j \equiv i \pmod{k}$ as follows: let $R^{(i,j)}$ be the subset of unit-height rectangles that are stabbed by one of the k consecutive horizontal lines $y = j, j + 1, \ldots, j+k-1$ but not by the line $y = j+k$ (see figure 3.2). The horizontal

lines are chosen to be unit distance apart because of the height of the rectangles.

(a) Solve the problem exactly for each $R^{(i,j)}$ by the algorithm in lemma 3.1. Let $C^{(i,j)}$ be the number of stabbing points returned for the subset $R^{(i,j)}$.

(b) Set $C^{(i)} \leftarrow \sum_j C^{(i,j)}$.

2. Return the minimum C among $C^{(0)}, C^{(1)}, \ldots C^{(k-1)}$.

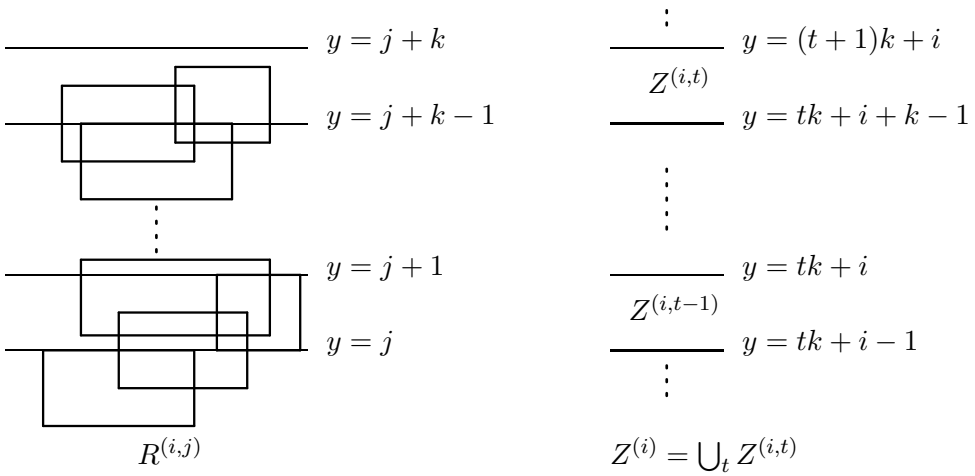


Figure 3.2: Illustration of $R^{(i,j)}$ and $Z^{(i,t)}$

Let Z denote the set of stabbing points in an optimal solution. For an integer t, let $Z^{(i,t)}$ denote the set of stabbing points in that optimal solution that lie in the semi-open strip $tk + i + k - 1 \leq y < (t+1)k + i$ (see figure 3.2). By this definition of $Z^{(i,t)}$, we have $Z^{(i,t_1)} \cap Z^{(i,t_2)} = \emptyset$ whenever $t_1 \neq t_2$. Let $Z^{(i)} = \bigcup_t Z^{(i,t)}$. Consequently

$$|Z^{(i)}| = \sum_t |Z^{(i,t)}|. \tag{3.1}$$

Let $X^{(i,j)}$ be the number of points in Z that stab the rectangles in $R^{(i,j)}$. Then $C^{(i,j)} \leq X^{(i,j)}$ since $C^{(i,j)}$ is the optimal number of points stabbing all rectangles in

$R^{(i,j)}$, and therefore

$$C^{(i)} \;=\; \sum_j C^{(i,j)} \;\leq\; \sum_j X^{(i,j)}. \tag{3.2}$$

For a fixed i, the points in $Z^{(i,t)}$ are counted at most twice in $\sum_j X^{(i,j)}$ and the rest of the points in Z are counted once. Thus

$$\sum_j X^{(i,j)} \;\leq\; |Z| + \sum_t |Z^{(i,t)}|. \tag{3.3}$$

From (3.2), (3.3) and (3.1) we have

$$C^{(i)} \;\leq\; |Z| + \sum_t |Z^{(i,t)}| \;\leq\; |Z| + |Z^{(i)}| \tag{3.4}$$

It can also be seen that $Z^{(i_1)} \cap Z^{(i_2)} = \emptyset$ whenever $i_1 \neq i_2$, and $Z = \bigcup_{i=0}^{k-1} Z^{(i)}$. This implies that

$$\sum_{i=0}^{k-1} (|Z| + |Z^{(i)}|) \;\leq\; (k+1)|Z|. \tag{3.5}$$

Therefore

$$\begin{aligned} C = \min_{i=0,\ldots,k-1} C^{(i)} \;&\leq\; \frac{1}{k} \sum_{i=0}^{k-1} C^{(i)} \\ &\leq\; \frac{1}{k} \sum_{i=0}^{k-1} (|Z| + |Z^{(i)}|) \quad \text{using (3.4)} \\ &\leq\; \left(1 + \frac{1}{k}\right) |Z| \qquad \text{using (3.5)}. \end{aligned}$$

Sorting the rectangles can be done in a preprocessing step. For a fixed i, running the exact algorithm for all the $R^{(i,j)}$ subsets will require $O(k\Delta n 2^{2k\Delta})$ time. As the process is repeated for k possible values of i, the overall running time for the approximation scheme is therefore $O(n \log n + k^2 \Delta n 2^{2k\Delta})$. □

3.2 PTAS for Almost Squares

A PTAS for covering a set of points in the plane with squares is already known and this translates to a PTAS for piercing a set of squares with points.

Theorem 3.2 (Feder and Greene, Gonzalez) *For $\epsilon > 0$, the piercing number of a set of n axis-parallel unit squares can be approximated within a factor of $(1+\epsilon)$ in $n^{O(1/\epsilon)}$ time.*

Corollary 3.2.1 *An ϵ-approximation of the cardinality of the minimum piercing set of a collection of uniform-sized axis-parallel rectangles can be computed in $n^{O(1/\epsilon)}$ time.*

Proof: By appropriate scaling, the set of rectangles can be turned into a set of unit squares while preserving the intersection graph (see section 1.1). By theorem 3.2, an ϵ-approximation of the piercing set of the transformed set of squares can be computed in $n^{O(1/\epsilon)}$ time. □

Although the algorithms make use of dynamic programming, it is not obvious how those algorithms can be adapted for unit-height rectangles with bounded width. Our approach for a depth-restricted set of unit-height rectangles, however, can be utilized for a set of unit-height rectangles that have width between constant lower and upper bounds. By appropriate scaling, the bounds on the widths of the rectangles can be mapped to the range $1 \leq \mathit{width} \leq \alpha$, where α is the upper-to-lower-bound ratio, without modifying the intersection graph. We also make the following observations regarding the minimum piercing sets of such rectangles.

Observation 3.1 *In an optimal solution of the piercing problem in the case of a set of axis-parallel squares, there can be at most four stabbing points within an axis-parallel unit square window.*

Proof: For contradiction, let there be an optimal solution which contains a set

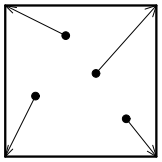


Figure 3.3: Unit square window containing piercing points

of more than four points that fit within a unit square window. Since every axis-parallel square intersecting the window contains a corner point of the window, that set of points can be replaced by the four corner points of the unit square window without leaving any square unstabbed (see figure 3.3). The replacement produces a new solution which has fewer points than the original solution and thus contradicts the assumption that the original solution is optimal. □

Observation 3.2 *Let $\alpha \geq 1$ be a constant. In an optimal solution for the piercing problem in the case of a set of unit-height axis-parallel rectangles with $1 \leq$ width $\leq \alpha$, there can be at most $2(1 + \lceil \alpha \rceil)$ stabbing points within a unit-height window of width α.*

Proof: A window of unit height and α width can have $\lceil \alpha \rceil - 1$ adjacent unit squares and the remaining space is a rectangular area of unit height and at most unit width [figure 3.4]. When any unit-height rectangle with $1 \leq$ *width* $\leq \alpha$ intersects this window, the rectangle intersects at least one of the squares or the remaining rectangle. By observation 3.1, any of the squares (or the remaining rectangle)

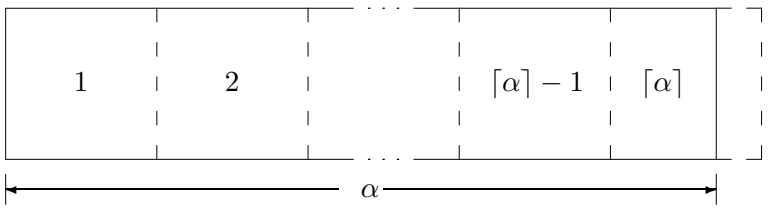


Figure 3.4: Rectangular unit-height window of width α

can contain at most four stabbing points. These internal stabbing points can be replaced by the corner points of a square (or the remaining rectangle) without leaving any rectangle unstabbed. Applying the replacement to all such squares (or the remaining rectangle) imply that there can be at most $2(1+\lceil\alpha\rceil)$ stabbing points within the window with at most $1+\lceil\alpha\rceil$ stabbing points lying on each horizontal line. □

Lemma 3.2 *Let $\alpha \geq 1$ be a constant and fix an integer $k \geq 1$. If a set of unit-height axis-parallel rectangles with $1 \leq$ width $\leq \alpha$ can be stabbed using k horizontal lines, then the cardinality of its minimum piercing set can be computed exactly in $O\left(n^{2k(1+\lceil\alpha\rceil)+1}\right)$ time.*

Proof: Let $R = \{R_1, R_2, R_3, \ldots, R_n\}$ be a set of unit-height axis-parallel rectangles with $1 \leq$ *width* $\leq \alpha$ that can be stabbed by k horizontal lines. Let $\{a_1, a_2, \ldots, a_m\}$ be the sorted set of x-coordinates of the corner points of the rectangles in R with $a_1 < a_2 < \cdots < a_m$. Let P_i be the set of candidate stabbing points, *i.e.*, corners and intersection points of the squares, on the line $x = a_i$.

We can extend observation 3.2 for a vertical strip of width α when the squares can be stabbed by k horizontal lines. In this case there can be at most $k(1+\lceil\alpha\rceil)$ points lying within the strip. Let W_i be the vertical strip of width α with the left side of the strip on $x = a_i$. A dynamic programming table can be created for each

vertical line $x = a_i$ and each subset of at most $k(1 + \lceil\alpha\rceil)$ stabbing points lying within a vertical strip of width α.

Let S_i be the set of rectangles that are intersected by the line $x = a_i$. Also for set of candidate stabbing points P, let $\mathbb{S}_P$ be the set of rectangles stabbed by the points in P, and let C_S be the minimum number of points needed to stab a set S of rectangles. The dynamic table entry $A[i, P]$, defined for each vertical line $x = a_i$ and each set of $k(1 + \lceil\alpha\rceil)$ stabbing points lying within a vertical strip of width α, indicates the cardinality of an optimal set of points $X_{(i,P)}$ such that

i) $X_{(i,P)}$ stabs all rectangles having their left endpoint on or to the left of the vertical line $x = a_i$ except the rectangles stabbed by any point in P, and

ii) there are at most $k(1 + \lceil\alpha\rceil)$ stabbing points within a vertical strip of width α for $P \cup X_{(i,P)}$.

The necessary entries of table A can be filled up as follows:

1. Base case: For each set of candidate stabbing points P lying within W_1 and with $|P| \leq k(1 + \lceil\alpha\rceil)$, set

$$A[1, P] \leftarrow C_{S_1 \setminus \mathbb{S}_P}$$

2. For $i \leftarrow 2$ to m do

   For each set of candidate stabbing points P lying within W_i and with $|P| \leq k(1 + \lceil\alpha\rceil)$, set $A[i, P]$ to

$$\min_{P'}(A[i-1, (P \cup P') \cap W_{i-1}] + C_{\mathbb{S}_{P'} \setminus \mathbb{S}_P} + C_{(S_i \setminus S_{i-1}) \setminus \mathbb{S}_P})$$

   where the minimum is taken over the subsets $P' \subseteq P_{i-1}$ such that $|P'| \leq k$ and $|(P \cup P') \cap W_{i-1}| \leq k(1 + \lceil\alpha\rceil)$. The term $A[i-1, (P \cup P') \cap W_{i-1}]$ is

a previously computed result in the table where the window containing the points excluded from piercing rectangles is shifted to the left and aligned with the previous *event line*. The term $C_{\mathbb{S}_{P'} \setminus \mathbb{S}_P}$ accounts for the rectangles omitted due to the inclusion of P' in the restricted set of points. Finally the term $C_{(S_i \setminus S_{i-1}) \setminus \mathbb{S}_P}$ corresponds to the minimum number of piercing number for the additional rectangles beginning on the line $x = a_i$ and not pierced by any point in P.

3. Return $A[m, \emptyset]$.

The algorithm is similar to that in lemma 3.1, but here we place a bound on the number of piercing points within a constant sized window instead of restricting the size of rectangle sets.

Enumerating the candidate stabbing points takes $O(n^2)$ time. The points are sorted based on x-coordinates in $O(n^2 \log n)$ time as part of preprocessing. The computation of P_i and W_i can also be done in the preprocessing and takes $O(n^2)$ time. Unlike the algorithm in lemma 3.1, the C values are computed only for the necessary sets determined during the execution of the algorithm. However, since any of such sets can be stabbed by a vertical line and at most n intervals formed by the squares on a vertical line need to be sorted, the computation will take $O(n \log n)$ time using the method of section 1.4. Exclusion of a set of rectangles stabbed by at most $k(1+\lceil \alpha \rceil)$ points takes $O(kn)$ time. Therefore the time required for generating the additive C term in the recursive formula is $O(kn) + O(n \log n)$. Generating all subsets of size j from a set of n points takes $O\left(\binom{n}{j}\right) = O(n^j)$ time. Therefore for a fixed value of i, generating appropriate P and P' sets and evaluating the recursive definition take $O\left(\sum_{j=0}^{k} O\left(n^j\right) \left[O\left(\left(n^2\right)^{k(1+\lceil \alpha \rceil)-j}\right) + O(n \log n) + O(kn)\right]\right) = O\left(\sum_{j=0}^{k} n^{2k(1+\lceil \alpha \rceil)-j}\right) = O\left(n^{2k(1+\lceil \alpha \rceil)}\right)$ time. With m being $O(n)$, step 2 requires

$O\left(n^{2k(1+\lceil\alpha\rceil)+1}\right)$ time, which in turn is the running time of the algorithm. This time complexity subsumes the time required for other preprocessing. □

Theorem 3.3 *Let $\alpha \geq 1$ be a constant and fix an integer $k \geq 1$. If a set of unit-height axis-parallel rectangles with $1 \leq$ width $\leq \alpha$ can be stabbed using k horizontal lines, then the cardinality of its minimum piercing set can be approximated within a factor of $(1 + \frac{1}{k})$ in $O\left(kn^{2k(1+\lceil\alpha\rceil)+1}\right)$ time.*

Proof: The approximation is done by running the algorithm in theorem 3.1 with the following exception: the algorithm used for solving the subproblem is the algorithm in lemma 3.2. The proof of correctness of the approximation in theorem 3.1 holds even with this modification. The running time is however $O\left(kn^{2k(1+\lceil\alpha\rceil)+1}\right)$ since an iteration over the entire set of rectangles takes at most $O\left(n^{2k(1+\lceil\alpha\rceil)+1}\right)$ time. □

Our ϵ-approximation scheme for bounded width unit-height rectangles thus takes $n^{O(1/\epsilon)}$ time. Chan's algorithm for piercing fat objects can be applied to such rectangles; however, the algorithm still requires $n^{O(1/\epsilon^2)}$ time [Cha03].

3.3 PTAS for General Unit-Height Case

Designing an approximation scheme for unit-height rectangles of arbitrary width that takes running time comparable to that for bounded width is still difficult. The reason is that arbitrarily long, *i.e.*, *thin* rectangles require that the algorithm must "remember" a lot more information for making an optimal choice. Therefore we propose a two-step approximation, the first one along the horizontal direction

by using incremental division of rectangles and the second one along the vertical direction using the shifting technique. In order to be able to divide the rectangles in the first step, we make use of a linear-time factor 2 approximation algorithm described in lemma 3.3.

Lemma 3.3 *For a set of n unit-height axis-parallel rectangles, sorted according to their right boundaries, a factor 2 approximation of its piercing set can be computed in $O(n)$ time.*

Proof: The algorithm for factor 2 approximation simply uses the linear time solution for one-dimensional case (see section 1.4):

1. Consider all horizontal lines $x = i$ that intersect at least one rectangle in the set, where i is an integer. On each of these horizontal lines, compute in linear time the piercing set of the one-dimensional intervals formed by the rectangles intersecting that line. Let $P^{(i)}$ be the solution for the line $x = i$.

2. Return $P = \bigcup_i P^{(i)}$.

Since each rectangle can be intersected by at most two of the horizontal lines, the overall running time of the algorithm is $O(n)$.

Let $P_{\text{odd}} = \bigcup_{i:\ i \text{ odd}} P^{(i)}$ and $P_{\text{even}} = \bigcup_{i:\ i \text{ even}} P^{(i)}$, and also let Z be a piercing set for the entire set of rectangles. By this definition, $P = P_{\text{odd}} \cup P_{\text{even}}$ and $P_{\text{odd}} \cap P_{\text{even}} = \emptyset$, *i.e.*, $|P| = |P_{\text{odd}}| + |P_{\text{even}}|$. Let $\mathcal{R}_{\text{odd}}$ be the set of rectangles that are stabbed by P_{odd} and $\mathcal{R}_{\text{even}}$ be the set of rectangles stabbed by P_{even}. Let $\mathcal{R}_j$ be the maximum set of rectangles in $\mathcal{R}_{\text{odd}}$ intersected by a line $x = j$, where j is odd. Because of unit height, any rectangle in $\mathcal{R}_j$ does not intersect any other rectangle in $\mathcal{R}_{\text{odd}} \backslash \mathcal{R}_j$. A similar argument holds for rectangles in $\mathcal{R}_{\text{even}}$. This

property of disjointedness and the optimality of local solutions imply that $|P_{\text{odd}}|$ is a minimum piercing set for $|\mathcal{R}_{\text{odd}}|$, and thus $|P_{\text{odd}}| \leq |Z|$ and $|P_{\text{even}}| \leq |Z|$. Therefore $|P| = |P_{\text{odd}}| + |P_{\text{even}}| \leq 2|Z|$. □

We can divide a subregion or horizontal slab by vertical boundaries and add additional piercing points along the boundaries. Application of lemma 3.3 to the rectangles within a window returns at most twice the piercing number of the rectangles. Inversely, we need at least half of the number of points returned by lemma 3.3 to pierce the rectangles, and thus obtain a lower bound on the piercing number of the rectangles. By careful placement of the boundaries based on the lower bounds, it is possible to approximate the piercing number of rectangles in a horizontal slab in polynomial time. Lemma 3.4 describes the strategy in detail.

Lemma 3.4 *For $k' > 0$ and an integer $k \geq 1$, if a set of n unit-height axis-parallel rectangles can be stabbed by k horizontal lines, its piercing set can be approximated within a factor of $\left(1 + \frac{1}{k'}\right)$ in $O((2\lceil kk' \rceil + k - 1)n^{4\lceil kk' \rceil + 2k - 1})$ time.*

Proof: Let $R = \{R_1, R_2, R_3, \ldots, R_n\}$ be a set of axis-parallel unit-height rectangles that can be stabbed by k horizontal lines. Let the sorted set of x-coordinates of the corners of the rectangles in R be $\{a_1, a_2, a_3, \ldots, a_m\}$ with $a_1 < a_2 < \cdots < a_m$. An approximation of the piercing set of R can be computed as follows:

1. Set $S \leftarrow R$, $P \leftarrow \emptyset$, $i \leftarrow 0$.

2. Repeat

   Set $i \leftarrow i + 1$.

   Let $S_i \subseteq S$ be the set of rectangles intersected by the vertical line $x = a_i$.

   Also let $S'_i \subseteq S$ be the set of rectangles that have their right endpoint to

the left of the line $x = a_i$. Compute a lower bound of the piercing set of S'_i using the algorithm in lemma 3.3. If the lower bound is at least $\lceil kk' \rceil$ or if $i = m$, then

(a) Compute the piercing set P_i of S_i in linear time using the intervals formed by rectangles in S_i on the vertical line $x = a_i$.

(b) Find the exact piercing set P'_i of S'_i using exhaustive search.

(c) Set $P \leftarrow P \cup P_i \cup P'_i$, $S \leftarrow S \backslash (S_i \cup S'_i)$.

while $S \neq \emptyset$.

3. Return P.

Let $\mathcal{I}$ be the set of values of i for which steps 2(a)-2(c) are executed. The algorithm in effect partitions R into sets S_i and S'_i, $i \in \mathcal{I}$. Since $m \in \mathcal{I}$, every rectangle in R is considered. Therefore each rectangle in R is considered exactly once for stabbing. This justifies correctness. Furthermore, for $i, j \in \mathcal{I}$ and $i \neq j$, no rectangle in S'_i overlaps or intersects with any rectangle in S'_j. Therefore $P'_i \cap P'_j = \emptyset$ for $i \neq j$ and $i, j \in \mathcal{I}$. Also $P_i \cap P'_i = \emptyset$, since all points in P_i lie on the line $x = a_i$ whereas none of the points in P'_i lie on that line. For $i, j \in \mathcal{I}$ and $i \neq j$, it is also the case that $P_i \cap P_j = \emptyset$ as the sets lie on different vertical lines $x = a_i$ and $x = a_j$. Thus the sets P_i and P'_i, $i \in \mathcal{I}$, form a partition of P and hence

$$|P| = \sum_{i \in \mathcal{I}} (|P_i| + |P'_i|). \tag{3.6}$$

Let Z be a piercing set of R. Also let $Z'_i \subseteq Z$ be the set of points stabbing the rectangles in S'_i. The disjointedness property of P'_i sets also hold for Z'_i sets and therefore

$$\sum_{i \in \mathcal{I}} |Z'_i| \ \leq \ |Z|. \tag{3.7}$$

Evidently $|P'_i| \leq |Z'_i|$ as P'_i is a piercing set of S'_i. Using this fact and (3.7), we have

$$\sum_{i\in\mathcal{I}} |P'_i| \;\leq\; \sum_{i\in\mathcal{I}} |Z'_i| \;\leq\; |Z|. \tag{3.8}$$

We can now calculate a bound for the approximation ratio:

$$\begin{aligned}
\frac{|P|}{|Z|} &= \frac{\sum_{i\in\mathcal{I}} |P'_i|}{|Z|} + \frac{\sum_{i\in\mathcal{I}} |P_i|}{|Z|} && \text{from (3.6)} \\
&\leq 1 + \frac{\sum_{i\in\mathcal{I}} |P_i|}{|Z|} && \text{using (3.8)} \\
&\leq 1 + \frac{\sum_{i\in\mathcal{I}} |P_i|}{\sum_{i\in\mathcal{I}} |P'_i|} && \text{using (3.8) again} \qquad (3.9) \\
&\leq 1 + \frac{\max_{i\in\mathcal{I}\setminus\{m\}} |P_i|}{\min_{i\in\mathcal{I}\setminus\{m\}} |P'_i|}\,. && (3.10)
\end{aligned}$$

In (3.10), we can ignore the case where $i = m$ because for that value of i, $|P_i| = 0$, and consequently the bound in (3.9) does not exceed the bound in (3.10) with that particular exclusion. Also we shall consider only the cases where $\mathcal{I}\setminus\{m\} \neq \emptyset$, because in the other cases the whole set of rectangles requires less than $\lceil kk' \rceil$ stabbing points that are computed exactly. Therefore, without loss of generality, we get $\min_{i\in\mathcal{I}\setminus\{m\}} |P'_i| \geq \lceil kk' \rceil$. Also $\max_{i\in\mathcal{I}\setminus\{m\}} |P_i| \leq k$, since the rectangles can be stabbed by k horizontal lines. Using (3.10) and these bounds, we get:

$$\begin{aligned}
|P| &\leq \left(1 + \frac{k}{\lceil kk' \rceil}\right) |Z| \\
&\leq \left(1 + \frac{1}{k'}\right) |Z|,
\end{aligned}$$

i.e., P is a $\left(1 + \frac{1}{k}\right)$-factor approximation of Z.

In the preprocessing step, the rectangles need to be sorted based on their left endpoints and this takes $O(n \log n)$ time. At each iteration of step 2, the lower bound computation takes $O(|S'_i|)$ time. The overall time spent for this operation is $\sum_{i=1}^{m} O(|S'_i|) = O(n^2)$ as $m = O(n)$ and $|S'_i| \leq n$. For a brute force method, the

candidate points are the corner points and the intersection points. The number of these points is maximized if the rectangles intersect at points other than corners, and there can be at most two such points for a pair of rectangles. For S'_i, the maximum number of candidate points is therefore $2\binom{|S'_i|}{2} + 4|S'_i| = |S'_i|^2 + 3|S'_i|$. Checking whether j points stab the rectangles in S'_i takes $O(j|S'_i|)$ time without using any special data structure (like range trees). The number of stabbing points required to stab all rectangles in S'_i, $i \in \mathcal{I}$, can be at most $2\lceil kk' \rceil + k - 1$, because at each vertical line at most k additional stabbing points can appear and the algorithm uses a 2-approximation for determining whether $\lceil kk' \rceil$ points are needed. Therefore computation of P'_i, $i \in \mathcal{I}$, can take $O\left((2\lceil kk' \rceil + k - 1)|S'_i|\binom{|S'_i|^2 + 3|S'_i|}{2\lceil kk' \rceil + k - 1}\right) = O((2\lceil kk' \rceil + k - 1)S_i'^{4\lceil kk' \rceil + 2k - 1})$ time. Since $\sum_{i \in \mathcal{I}} |S'_i| \leq n$, computation of all P'_i, $i \in \mathcal{I}$, takes $O((2\lceil kk' \rceil + k - 1)n^{4\lceil kk' \rceil + 2k - 1})$ time. Also $\sum_{i \in \mathcal{I}} P_i \leq n$, and calculating the P_i's takes $O(n)$ time. The overall time required for the algorithm is therefore $O(n \log n + (2\lceil kk' \rceil + k - 1)n^{4\lceil kk' \rceil + 2k - 1}) = O((2\lceil kk' \rceil + k - 1)n^{4\lceil kk' \rceil + 2k - 1})$. □

Theorem 3.4 *An ϵ-approximation scheme for the piercing problem for a given a set of n unit-height axis-parallel rectangles takes $n^{O(1/\epsilon^2)}$ time.*

Proof: The approximation algorithm is similar to the algorithm in theorem 3.1:

1. Set $k \leftarrow \lceil \frac{3}{\epsilon} \rceil$, $k' \leftarrow k$.

2. For $i \leftarrow 0$ to $k - 1$ do

   Group rectangles into $R^{(i,j)}$'s in the same manner as in the algorithm in theorem 3.1. Find $P^{(i,j)}$, the $(1 + 1/k)$-factor approximation of piercing set, for each $R^{(i,j)}$ using the algorithm in lemma 3.4. Let $P^{(i)} = \bigcup_j P^{(i,j)}$.

3. Return the set P with minimum cardinality among $P^{(0)}, P^{(1)}, \ldots, P^{(k-1)}$.

Following the definition of Z and $Z^{(i)}$ in theorem 3.1, it can be shown that

$$|P^{(i)}| \leq \left(1 + \frac{1}{k}\right)(|Z| + |Z^{(i)}|)$$

and therefore

$$|P| \leq \left(1 + \frac{1}{k}\right)^2 |Z|.$$

Since $k = \lceil \frac{3}{\epsilon} \rceil \geq 1$, $\frac{1}{k} \geq \frac{1}{k^2}$. Hence we have

$$\begin{aligned} |P| &\leq \left(1 + \frac{2}{k} + \frac{1}{k^2}\right)|Z| \\ &\leq \left(1 + \frac{3}{k}\right)|Z| \\ &= \left(1 + \frac{3}{\lceil 3/\epsilon \rceil}\right)|Z| \\ &\leq (1+\epsilon)|Z|. \end{aligned}$$

Each iteration of step 2 takes $O((2k^2 + k - 1)n^{4k^2+2k-1})$ time and the overall running time of the algorithm is $O(k(2k^2 + k - 1)n^{4k^2+2k-1})$. As $k = O(1/\epsilon)$, the running time of the ϵ-approximation scheme becomes $n^{O(1/\epsilon^2)}$. □

Chapter 4

Dynamic Piercing

The assumption behind all the approximation schemes (see section 2.2) for rectangle piercing presented in the previous chapter is that complete information regarding the entire set of rectangles is available beforehand and is randomly accessible. This type of algorithms are regarded as *static*. It is natural to seek algorithms for the *dynamic* version of the problem. In a dynamic setting, rectangles can be added or removed from the set and the modification to the set must take place according to the order of arrival of request for modification. The approximate piercing number (see section 1.1) for the modified set is reported at each modification. It is desired that the time required for updating the approximate piercing number for a set of n rectangles be within a factor of $\frac{1}{n}$ of the running time for the static algorithm. But the running time of the approximation schemes for the static case is rather large ($n^{O(1/\epsilon^2)}$). Consideration of a dynamic ϵ-approximation algorithm (or scheme) seems to be unappealing with that much running time. However, a factor 2 approximation algorithm takes $O(n \log n)$ time in the static case, taking into account the time required for sorting the rectangles. Achieving an $O(\log n)$ update

time for the dynamic rectangle piercing problem is not a trivial task. The problem of dynamically maintaining a factor 2 approximation of the piercing number for a set of unit-height rectangles reduces to the dynamic interval piercing problem in the same way as the static case does in lemma 3.3. Interval piercing is interesting in its own right, since it is equivalent to finding maximum independent set (see section 1.2) in *interval graphs*, *i.e.*, intersection graphs (see section 1.1) for intervals. In other words, the piercing number equals the packing number (see section 1.2) in one dimension. Finding the maximum independent set in an interval graph is often studied as the *activity selection* problem [CLRS01]. In this chapter we provide an algorithm for interval piercing problem in a non-static setting with an effort to make the update time comparable to $O(\log n)$.

4.1 Dynamic Interval Piercing

Katz *et al.* provide a data structure for maintaining a minimum piercing set of intervals under insertion and deletion of intervals [KNM03]. They also mention several applications of dynamic piercing. The algorithm begins with a number of initial intervals and allows addition and deletion of intervals. If the number of intervals does not exceed n after any insertion of intervals, then their data structure can report the minimum piercing set and consequently the piercing number in $O(p \log n)$ time, where p is the piercing number of the updated set of intervals. It should be noted that without any kind of restriction on the intervals, p can be $O(n)$. Their algorithm first computes a *right-to-left piercing set* of the intervals using the algorithm of section 1.4, but in the right to left direction as outlined at the end of that section. They then construct a data structure of $O(n)$ size using a cascade of balanced binary search trees (BST) (see figure 4.1). The cascading scheme works

as follows:

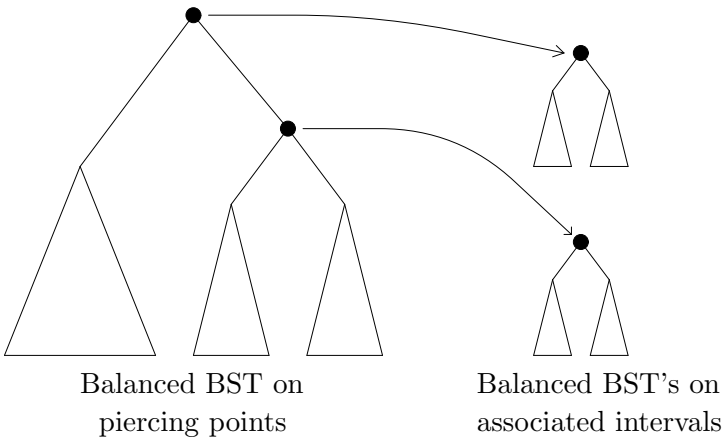


Figure 4.1: Data structure for dynamic piercing

The first balanced binary search tree is constructed for the piercing points of the initial intervals with each node corresponding to a separate piercing point. For each node of this tree a separate balanced binary search tree is maintained. An auxiliary tree is constructed using the right endpoints of the intervals pierced by the point corresponding to a node. Each node in the auxiliary tree also stores information about the rightmost left endpoint of the intervals represented within the subtree rooted at that node. The construction process take $O(n \log n)$ time.

When a new interval arrives and it does not cause the piercing set to change, only an auxiliary tree needs to be updated. The auxiliary tree is associated with the node representing the piercing point that intersects the interval. This update can be done in $O(\log n)$ time. When the newly arrived interval is not pierced by any of the current piercing points, several changes may occur to the primary search tree besides related auxiliary trees. Moreover, a new piercing point may cause a sequence of updates as some of the old piercing points may change. The number of updates required for the insertion of an interval is bounded by the size of the new minimum piercing set. With each change in the piercing set, adjustments in the

trees take $O(\log n)$ time and a sequence of at most p changes to the piercing set (due to insertion of a single interval) needs $O(p \log n)$ time in the worst case. Therefore the time spent for insertion of an interval is $O(p \log n)$, where p is the updated piercing number. Similarly in the case of deletions, when the left endpoint of the interval is not a piercing point, changes to auxiliary trees suffice and take $O(\log n)$ time. Whereas in other cases, a sequence of updates to the relevant tree structures may be necessary and thus need $O(p \log n)$ time with p being the updated piercing number. Katz *et al.* claim the bound to be $O(p \log \frac{n}{p})$ by application of Hölder's inequality.

Theorem 4.1 (Katz *et al.*) *For a set of intervals, the size of which never exceeds n, it is possible to construct a data structure of size $O(n)$ in $O(n \log n)$ time such that the minimum piercing set of the intervals (or the maximum independent set of the interval graph) can be maintained under insertion and deletion of intervals from the set in $O(p \log \frac{n}{p})$ time per update, where p is the updated piercing number (or the size of the maximum independent set).*

4.2 Incremental Piercing in $O(\log n)$ Update Time

We consider the incremental or insertion-only scenario for intervals with the intent of improving the update time. We provide a data structure that can efficiently maintain the piercing set for intervals under insertions. Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of n intervals with $I_k = [l_k, r_k]$, sorted by their right endpoints r_k. In addition, we consider two *sentinel* intervals $I_0 = [-\infty, -\infty]$ and $I_\infty = [\infty, \infty]$ for convenience. It is interesting that the relationship between intervals used for the incremental approach stems actually from the strategy of selecting piercing points by the greedy

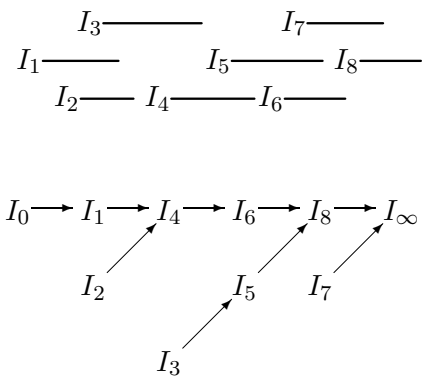


Figure 4.2: A set of intervals and corresponding graph T

method. We notice that if r_i and r_j $(i < j)$ are two consecutive piercing points chosen by the greedy algorithm then $r_j = \min_k\{r_k \mid l_k > r_i\}$. This property motivates the following definition (also used previously by Langerman [Lan00]):

Definition 4.1 *Let* $NEXT(I_i) = I_j$ *if* $r_j = \min_k\{r_k \mid l_k > r_i\}$.

It is possible to form a directed graph T with vertices $\mathcal{I} \cup \{I_0, I_\infty\}$ and edges $\{(I_i, I_j) \mid NEXT(I_i) = I_j\}$. Noticeably, T is acyclic and all vertices except I_∞ have out-degree one (see figure 4.2). Therefore T is a rooted tree and the piercing number corresponds to the length of the path from I_0 to I_∞. A very simple solution for maintaining the piercing set of the intervals dynamically is to maintain T in a data structure for *dynamic trees* [ST83] capable of answering path-length queries. The path-length query in such a data structure may take $\Omega(\log n)$ time in the worst case for a single edge update. However, for an n-vertex tree, the insertion of a single interval can cause as many as $\Omega(n)$ edge changes in T. Therefore we maintain a modified tree T' instead.

We need to define two subroutines before we can describe T':

(a) Given an interval I_i, compute $NEXT(I_i)$:

We need to determine k that minimizes r_k such that $l_k > r_i$. We can perform this search in logarithmic time using *priority search trees* [McC85], where intervals are treated as 2-dimensional points. In a priority search tree, one of the endpoints of the intervals (or the corresponding coordinate) is used for ordering while the other endpoint (or coordinate) defines priority. For our purpose, we store the intervals I_k in priority search trees ordered by l_k, with priorities defined by r_k. The root of any subtree in this structure corresponds to the interval with the median l_k value among all intervals rooted in that subtree. The rest of the intervals are then divided according to their l_k values so that intervals in one subtree have l_k values larger than the other subtree. At each node, the minimum r_k value over all intervals in the subtree rooted at that node is maintained as well. This data structure can be maintained in logarithmic time per update and the query for minimum priority value can be answered within the same time bound [McC85].

(b) Given an interval I_j, identify all I_i's such that $NEXT(I_i) = I_j$:

This is actually the "opposite" query. In order to identify the intervals we observe that

$$\begin{aligned} NEXT(I_i) = I_j &\iff l_j > r_i \text{ and } \forall k, l_k > r_i \Rightarrow r_j \leq r_k \\ &\iff l_j > r_i \text{ and } \forall k, r_j > r_k \Rightarrow l_k \leq r_i \\ &\iff l_j > r_i \geq \max_k \{l_k \mid r_k < r_j\}. \end{aligned}$$

When all intervals are sorted by right endpoints, all such I_i's appear in consecutive order. Therefore we can determine the first and last of these intervals in logarithmic time by means of another priority search tree. Unlike the tree in subroutine (a), the intervals I_k in this tree are ordered by r_k with priorities defined by l_k, and at each node, the maximum of the priority values is

maintained instead of the minimum one.

We define a *block* to be a maximal set of intervals with the same $NEXT$ value. We observe that blocks are disjoint. A more important observation, as shown in subroutine (b), is that elements within a block are consecutive. We can now describe the modified graph T'. Intervals are represented by vertices of the graph. We add an edge of weight zero between every pair of consecutive vertices in the same block. We place an edge of weight one from I_i to $NEXT(I_i)$ only when I_i is represented by the *last* vertex of the block. Even with this modification, T' remains a tree and sum of edge weights along a path in T' is the same as the distance between the end vertices in T. Therefore the piercing number corresponds to the total weight of the path from I_0 to I_∞ in T'. To be exact, the piercing number is one less than the weight of the path.

The original dynamic tree structure proposed by Sleator and Tarjan can maintain T' under edge insertions and deletions (by *link* and *cut* operations) in logarithmic time [ST83]. The structure also supports certain queries like maximizing edge costs along a path within the same time bound. However, for our particular type of queries, *i.e.*, summing edge weights along a path, we can use Alstrup *et al.*'s top-tree structure [AHdLT03]. Inserting a new interval I can trigger various changes in T'. There can be at most one insertion of a weight-1 edge leaving the vertex for I (computable by (a)) and at most one insertion of a weight-1 edge entering the vertex for I (computable by (b)). In addition, there can be at most one deletion of a zero-weight edge caused by splitting of a block, as well as insertions of zero-weight edges and deletions of weight-1 edges caused by merging of blocks. For each interval insertion, the number of splits is bounded by a constant but the number of merges may be large. However, since the total number of merges is bounded by n plus the number of splits, the amortized number of edge changes remains $O(1)$. Further-

more, the location of the merges and splits can be determined in $O(\log n)$ time by having an extra balanced search tree for holding the blocks' boundaries. Thus the overall amortized time for maintaining T' is $O(\log n)$. Therefore we conclude:

Theorem 4.2 *In an insertion-only scenario, the piercing number of a set of n intervals can be maintained in $O(\log n)$ amortized time per insertion.*

Corollary 4.2.1 *In an insertion-only scenario, a factor 2 approximation of the piercing number of a set of n unit-height rectangles can be maintained in $O(\log n)$ amortized time per insertion.*

Proof: As in lemma 3.3, we consider intervals formed by rectangles on horizontal lines at integer coordinates only. For each such horizontal line, separate data structures are maintained. If a rectangle intersects a horizontal line already intersected by another rectangle, the new interval on that line is added to the associated data structures. Otherwise new data structures are created. Updates in associated data structures or setting up a new one takes $O(\log n)$ amortized time, since a rectangle can intersect at most two such horizontal lines. In order to efficiently locate the appropriate horizontal lines and data structures or determine the necessity of creating new ones, a balanced binary tree structure on the y-coordinates can be used. On arrival of a new rectangle, query and maintenance of such a structure can be done in $O(\log n)$ time. □

Chapter 5

Conclusion

Approximation algorithms trade-off the quality of the solution in favor of less computation time. Our approximation schemes for the cases with restrictions on depth of a point and on the width of the rectangles have a better running time compared to that in the general unit-height case. The schemes in the former cases make use of dynamic programming but the dynamic programming tables require a lot of memory. Moreover, the space requirement in these schemes grows when smaller approximation ratios are sought. On the other hand, the approach used for the general unit-height case is straightforward and does not require a lot of space. The difference in the running times raises a non-trivial question: is it possible to design a PTAS for axis-parallel unit-height rectangles that takes $n^{O(1/\epsilon)}$ time? The crucial part of the answer to this question lies in the design of a polynomial-time exact algorithm for the special case where rectangles are known to be pierceable by a constant number of lines (see lemma 3.4).

Since the complexity of the piercing problem does not allow fully polynomial-time approximation schemes, our schemes appear to be more of theoretical interest

from an implementation point of view. The design of constant factor approximation algorithms for piercing arbitrarily sized rectangles is a challenging open problem and interesting from both the theoretical and practical perspective.

The dynamic piercing problem is interesting as well. We have shown how to maintain the piercing set for intervals in $O(\log n)$ amortized time in an insertion-only scenario. The result leads to another non-trivial question: can we design a fully dynamic interval piercing algorithm that takes $O(\log^{O(1)} n)$ time per insertion or deletion? As the interval piercing problem is equivalent to the maximum independent set problem for interval graphs, a dynamic solution is likely to have more practical applications.

Bibliography

[AHdLT03] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Maintaining information in fully-dynamic trees with top trees. `http://arxiv.org/abs/cs/0310065`, 2003.

[AS98] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys*, 30(4):412–458, December 1998.

[AvKS98] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry Theory and Applications*, 11(3-4):209–218, December 1998.

[Bak94] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, January 1994.

[Cha03] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46(2):178–189, February 2003.

[Cha04] T. M. Chan. A note on maximum independent sets in rectangle intersection graphs. *Information Processing Letters*, 89(1):19–23, January 2004.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[CM05] T. M. Chan and A. Mahmood. Approximating the piercing number for unit-height rectangles. To appear in *Proceedings of the Seventeenth Canadian Conference on Computational Geometry*, 2005.

[EKNS00] A. Efrat, M.J. Katz, F. Nielsen, and M. Sharir. Dynamic data structures for fat objects and their applications. *Computational Geometry: Theory and Applications*, 15(4):215–227, April 2000.

[FG88] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 434–444, 1988.

[FPT80] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. The complexity of covering and packing in the plane and related intersection graph problems. Technical report 80-05-02, University of Washington, Department of Computer Science, May 1980.

[FPT81] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, June 1981.

[FW91] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 281–288, 1991.

[GIK02] D. R. Gaur, T. Ibaraki, and R. Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilin-

ear partitioning problem. *Journal of Algorithms*, 43(1):138–152, April 2002.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[Gon91] T. F. Gonzalez. Covering a set of points in multidimensional space. *Information Processing Letters*, 40(4):181–188, November 1991.

[HM85] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, January 1985.

[HM87] D. S. Hochbaum and W. Maass. Fast approximation algorithms for a nonconvex covering problem. *Journal of Algorithms*, 8(3):305–323, September 1987.

[HM91] R. Hassin and N. Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29–42, January 1991.

[KNM03] M. J. Katz, F. Nielsen, and Segal M. Maintenance of a piercing set for intervals with applications. *Algorithmica*, 36(1):59–73, February 2003.

[KS04] S. Kovaleva and F. C. R. Spieksma. Approximation of rectangle stabbing and interval stabbing problems. In *Proceedings of the Twelfth Annual European Symposium on Algorithms*, number 3221 in Lecture Notes in Computer Science, pages 426–435, 2004.

[Lan00] S. Langerman. On the shooter location problem: Maintaining dynamic

circular-arc graphs. In *Proceedings of the Twelfth Canadian Conference on Computational Geometry*, 2000.

[McC85] E. M. McCreight. Priority search trees. *SIAM Journal of Computing*, 14(2):257–276, May 1985.

[Nie00] F. Nielsen. Fast stabbing of boxes in high dimension. *Theoretical Computer Science*, 246(1-2):53–72, September 2000.

[ST83] D. E. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, June 1983.

[SW96] M. Sharir and E. Welzl. Rectilinear and polygonal p-piercing and p-center problems. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 122–132, 1996.

[Tan79] S. L. Tanimoto. Covering and indexing an image subset. In *Proceedings of IEEE Computer Society Conference on Pattern Recognition and Image Processing*, pages 239–245, 1979.

[TF80] S. L. Tanimoto and R. J. Fowler. Covering image subsets with patches. In *Proceedings of the Fifth International Conference on Pattern Recognition*, pages 835–839, 1980.