On the Optimal Binary Plane Partition for Sets of Isothetic Rectangles[*]

Fabrizio d'Amore[†] Paolo Giulio Franciosa[†]

Abstract

We present a Binary Space Partition algorithm for a set of disjoint isothetic rectangles. It recursively splits the set by means of isothetic cutting lines, until no two rectangles belong to the same portion of plane. Rectangles intersected by a cutting line are split. We show that given n rectangles it is always possible to build a Binary Space Partition with no empty regions by means of no more than $6n - 1$ cuts and giving an $O(n \log n)$ space and time algorithm for doing this.

We generalize and improve a previous result achieved on isothetic line segments by Paterson and Yao [3].

Keywords: Analysis of algorithms, computational geometry, binary space partitions, isothetic rectangles.

1 Introduction

Given a region R containing a set of objects, the Binary Space Partition (BSP) problem consists in recursively partitioning R by a hyperplane into two regions, continuing this process until each obtained region intersects at most a fixed number of objects [1]. The number of these regions is the *size* of the BSP.

If an object is intersected by a cutting hyperplane it is split into two disjoint objects whose union returns the intersected one.

In this paper we consider the problem of finding a BSP of a set of n disjoint isothetic rectangles of $\mathcal{R}^2$, where *isothetic* means that each edge is parallel to a coordinate axis (for a more precise definition see [5, pages 321–322]). In [3] it was proved that for a set of n isothetic line segments of $\mathcal{R}^2$ a BSP always exists of size $4n$. In [4] a $O(n \log^2 n)$ time, $O(n \log n)$ space algorithm is given for finding a BSP of size $3n$. Although such algorithm can be applied to the problem of finding a BSP of a set of n disjoint rectangles (by considering the $4n$ line segments which constitute rectangle edges) it generally leads to BSPs with empty regions (i.e. intersecting no rectangle), and the size can grow up to $12n$.

In this paper we prove that a BSP of size $6n$ for n disjoint isothetic rectangles always exists, and give an $O(n \log n)$ algorithm for building the partition. We extend the technique used in [3] and improve the algorithm given in [4].

[*]Work partially supported by the ESPRIT II Basic Research Actions Program Project no. 3075 ("ALCOM") and by the Italian MURST National Project "Algoritmi, Modelli di Calcolo e Strutture Informative".

[†]Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", via Salaria 113, I-00198 Roma, Italia. E-mail: damore@disparcs.ing.uniroma1.it or francios@disparcs.ing.uniroma1.it

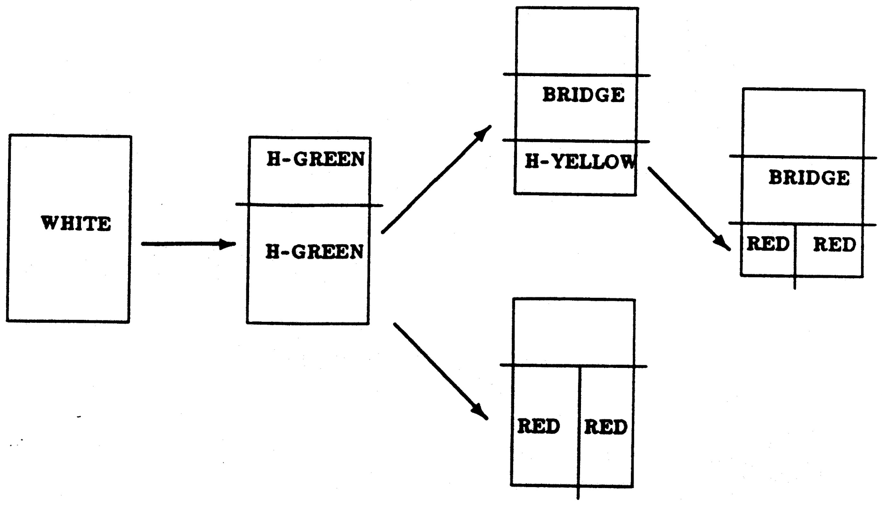

Figure 1: The decomposition of a rectangle.

2 Preliminaries

A *rectangle* is a closed interval of $\mathcal{R}^2$. A line λ and a rectangle Q are *adjacent* if and only if λ contains a side of Q. Moreover λ *cuts* Q if and only if λ intersects the interior of Q.

Any isothetic line which splits a given region into two (sub-)regions is called a *cutting line.*

A rectangle is *high-anchored* if it is the lower portion of a larger rectangle which has been horizontally split. Anchoring properties are also inherited when possible, in the sense that a rectangle is high-anchored also if it is the upper portion of a high-anchored rectangle that has been horizontally split. The *low-anchored*, *right-anchored* and *left-anchored* rectangles are analogously defined. For a region R we will denote these sets respectively by $\mathsf{high}(R)$, $\mathsf{low}(R)$, $\mathsf{right}(R)$ and $\mathsf{left}(R)$. Since a rectangle may be anchored on one or more edges the sets are not necessarily disjoint each other.

Now we introduce a coloring of rectangles, according to the following rules (see fig. 1):

- Rectangle Q is WHITE if and only if it is one of the original rectangles, i.e. it has never been split;
- Q is H-GREEN (V-GREEN) if and only if it is obtained by horizontally (vertically) cutting a WHITE rectangle;
- Q is a BRIDGE if and only if it is anchored on two opposite edges;
- Q is H-YELLOW (V-YELLOW) if and only if it is obtained by horizontally (vertically) cutting a H-GREEN (V-GREEN) rectangle and it is not a BRIDGE;
- Q is RED if and only if it is obtained by vertically (horizontally) cutting a H-GREEN or H-YELLOW (V-GREEN or V-YELLOW) rectangle.

A horizontal (vertical) edge of a region R is GREEN if and only if all the rectangles anchored on it are H-GREEN (V-GREEN).

3 An algorithm for the optimal plane partition

The algorithm finds a BSP of a region R by recursively cutting regions. It is assumed that R has a GREEN edge, and without loss of generality we suppose that it is on the right.

Algorithm: **Partition**

1) If R contains just one rectangle then return.
2) Let Q be the longest rectangle in $\mathsf{left}(R)$ or, if $\mathsf{left}(R)$ is empty, the rectangle in R having the leftmost right edge (break ties arbitrarily). Let ρ be the vertical line adjacent to the right edge of Q. Let R_l and R_r be the portion of R respectively to the left and to the right of ρ.
3) If there are no rectangles intersecting R_r then let σ be the horizontal line adjacent to Q creating no empty regions. Cut R by σ into R_1 and R_2. Partition (R_1). Partition (R_2). Return.
4) Cut R by ρ into R_l and R_r. Partition (R_r).
5) If Q is not the only rectangle intersecting R_l then let σ be the horizontal line adjacent to Q creating no empty regions: cut R_l by σ into R_1 and R_2. Partition (R_1). Partition (R_2).
6) Return.

The following lemmata show important properties of algorithm **Partition**.

Lemma 3.1 *Algorithm* **Partition** *is always invoked for partitioning regions with a* GREEN *edge.*

Proof. We will separately analyze the consequence of the various cuts carried out by the algorithm.

For the cuts at step 3, σ is a GREEN edge for both R_1 and R_2, since it does not cut any anchored rectangle.

With regard to the cuts at step 4, the right edge of R_r is GREEN since it coincides with the right edge of R. Note that since in this step no call for partitioning R_l is done, we do not need GREEN edges for R_l (if R_l needs to be partitioned this will be done in step 5).

Finally, for the cuts executed at step 5, we observe that σ is a GREEN edge for both R_1 and R_2, since it cuts no anchored rectangle. □

Lemma 3.2 *Algorithm* **Partition** *does not create empty regions.*

Proof. It directly derives from the algorithm. □

Lemma 3.3 *Algorithm* **Partition** *decomposes each original rectangle in at most 6 rectangles.*

Proof. We prove that a rectangle can only be decomposed as shown in fig. 1 (suppose without loss of generality that the first cut of a rectangle is horizontal). More precisely, RED and BRIDGE rectangles are never cut, and H-YELLOW (V-YELLOW) rectangles can only be vertically (horizontally) cut.

With reference to algorithm **Partition**, cuts carried out in steps 3 and 5 split no rectangle. On the contrary, cuts executed at step 4 can intersect rectangles. However it is worth observing that ρ can only intersect:

- WHITE rectangles;
- rectangles in right(R), which cannot be YELLOW or RED because the right edge of R is GREEN;
- at most two H-YELLOW or H-GREEN rectangles in high(R) $\cup$ low(R), which will be vertically cut. These two rectangles cannot be RED, otherwise they should be anchored on a GREEN edge of R.

It is immediate to see that vertical BRIDGEs cannot be cut and, since the right edge is GREEN, there is no horizontal BRIDGE in R. □

We can now show that a BSP with size not greater than $6n$ always exists.

Theorem 3.4 *Algorithm* **Partition** *partitions a region containing n rectangles in at most* $6n$ *regions, each containing exactly one rectangle.*

Proof. It directly derives from lemma 3.2 and lemma 3.3. □

4 Complexity Analysis

The algorithm makes use of some data structures, both static and dynamic. The former are useful for representing geometrical properties of the original set of rectangles, while the latter represent the sets of the anchored rectangles for the regions currently being partitioned.

More precisely, two static *layered segment trees* [6] representing the original rectangles are used. The *horizontal* one stores the horizontal edges according to their abscissae. Each node stores in a balanced data structure the pointers to the representations of the rectangles according to their ordinates. The *vertical* layered segment tree is analogously organized.

Moreover, the algorithm uses four static *layered range trees* [5, pages 84–85], in which the midpoints of respectively the left, right, high and low edges of the original rectangles are stored.

The only dynamic data structures are *priority search trees* [2]: for each region R there are four of them, representing sets left(R), right(R), high(R) and low(R).

The priority search tree for left(R) consists of a binary search tree whose leaves contain rectangles in left(R) sorted by ordinates; any internal node v also points to the leftmost rectangle contained in the subtree rooted at v that is not pointed to by any ancestor of v. These priority search trees are not complete binary trees, but their height is always not greater than $\lceil \log_2 n \rceil$. Priority search trees corresponding to high(R), low(R) and right(R) are analogously organized.

Without loss of generality we will refer in the sequel to a vertical cut which splits R into R_l and R_r.

The priority search trees for left(R_l) and right(R_r) are respectively those for left(R) and right(R), the trees for high(R_l) and high(R_r) are obtained by splitting the one for high(R), and those for low(R_l) and low(R_r) are obtained by splitting the one for low(R).

While splitting a priority search tree we do not need to rebalance the structure, because it suffices that its height do not grow. Hence the splitting process can be carried out in $O(\log n)$ time.

The trees for right(R_l) and left(R_r) are built by searching the horizontal layered segment tree for the rectangles intersected by the cutting line. The search process takes time $O(\log n + k)$, where k is the number of retrieved rectangles. It returns the rectangles as a sorted set, thus allowing the construction of the priority search trees in $O(k)$ time. These trees are built as complete binary trees.

We will now describe in detail how the structures are used while executing algorithm **Partition**. We will refer without loss of generality to the case in which the GREEN edge is on the right.

Step 1 can be performed by visiting the priority search trees associated to R until two rectangles are found (hence R contains more than one rectangle). If less than two rectangles are found, we need to visit one of the layered range trees in order to discover whether one or two WHITE rectangles are contained in R. This takes time $O(\log n)$.

Rectangle Q in step 2 is found in constant time on the priority search tree associated to $\mathsf{left}(R)$ or, if it is empty, it can be found in the layered range tree associated to the midpoints of the right rectangle edges. On this tree we start answering a orthogonal range query on R, but we stop as soon as the first point is found, thus spending $O(\log n)$ time.

Emptiness queries in steps 3 and 5 can be answered in $O(\log n)$ time by a process analogous to the one for testing the presence of just one rectangle in R.

Split operations in steps 3, 4 and 5 have already been described.

Theorem 4.1 *For a set of n disjoint isothetic rectangles, a Binary Space Partition of size $6n$ can be found in $O(n \log n)$ worst case time and space.*

Proof. The layered segment and range trees can be build in $O(n \log n)$.

The time complexity of one execution of algorithm **Partition**, not considering the recursive calls, is $O(\log n + k)$. Since the procedure is called at most $6n - 1$ times, and it is shown in lemma 3.3 that each original rectangle is split at most in 6 rectangles, the overall worst case time complexity is $O(n \log n)$.

The priority search trees corresponding to a region R need $O(r)$ space, where r is the number of rectangles in R, leading to overall $O(n)$ space. The layered range trees and the layered segment trees need $O(n \log n)$ space. □

References

[1] H. Fuchs, Z. Kedem and B. Naylor, On visible surface generation by a priori tree structures, *Computer Graphics* (SIGGRAPH '80 Conference Proceedings), 124–133.

[2] E.M. McCreight, Priority Search Trees, *SIAM J. Comput.* **14** (2) (1985) 257–276.

[3] M.S. Paterson and F.F. Yao, Binary Partitions with Applications to Hidden-Surface Removal and Solid Modelling, in: *Proc. 5th Annual Symposium on Computational Geometry* (1989), 23–32.

[4] M.S. Paterson and F.F. Yao, Optimal Binary Space Partitions for Orthogonal Objects, Research Report 158, Dept. of Computer Science, University of Warwick, 1990.

[5] F.P. Preparata and M.I. Shamos, *Computational Geometry*, (Springer-Verlag, New York, 1985).

[6] V. Vaishnavi and D. Wood, Rectilinear line segment intersection, layered segment trees, and dynamization, *J. Algorithms* **3** (1982) 160–176.