DIMACS Technical Report 2000-26
September 2000

On the Exact Size of the Binary Space Partitioning of Sets of Isothetic Rectangles with Applications

by

Piotr Berman[1]
Department of Computer Science
Pennsylvania State University
University Park, PA 16802
Email: berman@cse.psu.edu

Bhaskar DasGupta[2]
Department of Computer Science
Rutgers University
Camden, NJ 08102
Email: bhaskar@crab.rutgers.edu

S. Muthukrishnan
AT& T Labs – Research
180 Park Avenue
Florham Park, NJ 07932
Email: muthu@research.att.com

[1]Supported in part by NSF grant CCR-9700053 and by NLM grant LM05110.
[2]Supported in part by NSF Grants CCR 99-06106, CCR 91-19999 and CCR-9800086.

ABSTRACT

We show an upper bound of $3n$ on size of the Binary Space Partitioning (BSP) tree for a set of n isothetic rectangles, and an upper bound of $2n$ if the rectangles tile the underlying space. This improves the bound of $12n$ from [PY92] and $4n$ in [NW95, dAF92]. The BSP tree is one of the most popular data structures and even "small" factor improvements of $4/3$ or 2 we show improves the performance of applications relying on the BSP tree. Furthermore, our upper bounds yield improved approximation algorithms for several rectangular tiling problems in the literature. We also a show a lower bound of $2n$ in the worst case for a BSP for n isothetic rectangles, and a lower bound of $1.5n$ if they must form a tiling of the space.

1 Introduction

Binary Space Partitioning (BSP) for a collection of geometric objects on two dimensional plane (we restrict ourselves to two dimensional space throughout the paper) is defined as follows. The space is divided into two (not necessarily equal) parts by cutting objects with a line if necessary. Each fragment of the object belongs solely to one of the parts it falls in. The two resulting spaces are divided recursively; the process continues until at most one fragment of the original objects remains in any space.[1] This process can be naturally thought of as a binary tree (BSP tree) where a node represents a space and it stores the cut that splits the space into subspaces that its two children represent; each leaf of the BSP tree represents the final partitioning of the space and stores at most one fragment of an input object. Since a cut at some level may split an object into two, the number of regions in the final configuration, equivalently the number of leaves in the BSP tree, may exceed the number of input objects. Note that quadtrees, octtrees and grid files are all related to BSPs

Since the introduction of BSP trees in [FKN80], they have become one of the most popular data structures. They present a way to implement a geometric divide-and-conquer strategy. They have found numerous applications in: Graphics (hidden surface removal [D94], shadow generation [CF89]), Computational Geometry (ray-tracing [NT86], visibility problems [T92], solid geometry [TN87]), Robotics (motion planning [B93]), Spatial Databases [S90, van90] and Approximation Algorithms [KMP98, M90]). They are used in computer games such as DOOM and Quake, and there is a lot of publicly available code for different kinds of BSPs.

The fundamental parameter of interest about BSPs in all their applications is the size, that is, the number of leaves in a BSP tree. In two seminal papers, Paterson and Yao [PY90, PY92] established the first and essentially optimal bounds on the size of some BSPs. In two dimensions which is of our interest, they proved that any set of n line segments has a BSP of size $O(n \log n)$; for axis-parallel line segments, they proved that BSPs of $O(n)$ size exist. Some results are in [deGO] for more general class of objects on the plane. The problem of bounding the size of BSPs remains an active research area: e.g., in higher dimensions, for objects with bounded aspect ratio, etc. [NW95, AG+96]

In this paper, we focus on axis-parallel (isothetic) rectangles as do Paterson and Yao in [PY92]. Such rectangles form a very important class of objects in application domains because complex objects are often replaced by their bounding rectangles. They also arise naturally in plane tiling problems such as constructing two dimensional histograms, and as subproblems when higher dimensional hyperrectangles are projected onto two dimensions. Our focus is on proving bounds on the *exact* size of BSP trees in the worst case, that is, not just asymptotic size, but also the exact constants involved. In this paper, we prove the following results:

1. There exists a BSP tree of size at most $3n$ for collection of n isothetic rectangles. If the rectangles form a tiling (that is, they partition the underlying space), then we prove an improved upper bound of $2n$.

   Paterson and Yao proved an upper bound of $12n$ in [PY92]; subsequent improvements have led to the current best upper bound of $4n$ [NW95, dAF92].

2. We present first known nontrivial lower bounds on the size of a BSP tree for n isothetic rectangles in the worst case: we prove a lower bound of about $2n$. If the rectangles must form a tiling of the space, we show a lower bound of about $^3/_2 n$.

Size of the BSP trees determines the time and space of all applications reliant on them, improvements even by "small" factors ($^4/_3$ or 2) are highly desirable. Even construction time in securing these improvements is not a bottleneck because our upper bounds can be achieved by efficient algorithms taking $O(n \log n)$ time. It is a modification of the original algorithm in [PY92], matching it in its running time. The bulk of our technical achievement is our detailed amortized analyses of the algorithm. Finally, our result on the upper bound on the size of a BSP tree for rectangles that form a tiling leads to significant improvements on exiting algorithms for determining best rectangular tilings; what is the "best" is determined by different applications in spatial data structures, databases, parallel load balancing, video compression, etc.

In the rest of the paper, we first present the upper bounds (Section 2) followed by the lower bounds (Section 3). Both sections rely on geometric arguments and we clarify them mostly using figures, leaving formal description to

[1]The objects have to be disjoint since otherwise no BSP exists in which each final space contains at most one fragment of an object.

the final version. We also present an overview of how our upper bound leads to better approximation algorithms for determining best rectangular tilings, and their applications in Section 4.

2 Upper Bounds

Given a rectangular region **R** containing a set of n disjoint isothetic rectangles, the BSP problem we focus on consists of recursively partitioning **R** by a horizontal or vertical line into two subregions and continuing in this manner for each of the two subregions until each obtained region intersects at most one rectangle. If a rectangle is intersected by a cutting line into, it is then split into disjoint rectangles whose union is the intersected rectangle. The *size* of a BSP is the number of regions produced. Equivalently, a BSP of a set of rectangles contained in a rectangular region **R** is a binary tree, where each node is a rectangular subregion of **R**, each internal node is the union of its two children, the intersection of each leaf with the rectangles in our collection has at most one rectangle, and the root is the rectangular region **R**. The partition of **R** that corresponds to a BSP is the collection of the leaves of this tree, and the size of the BSP is the number of leaves in the tree. It is shown in [KMP98, MPS99] that a BSP of **R** with the minimum number of leaves can be computed using dynamic programming technique in $O(n^5)$ time.

Theorem 1 *We can compute a BSP of R containing at most $3n$ regions in $O(n \log n)$ time.*

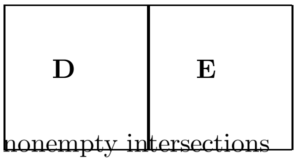
Restricting the set of rectangles in our collection to be a partition of **R** yields even better bounds on the size of the BSP.

Theorem 2 *Assume that the rectangles in our collection form a partition of R. Then, we can compute a BSP of R containing at most $2n$ regions in $O(n \log n)$ time.*

In the rest of this section, we will prove both theorems.

2.1 General Schema for the Proofs of Theorems 1 and 2

Both Theorems 1 and 2 uses the accounting method of amortized analysis for tighter bounds on the size of their respective BSPs. First, we describe the general schema and then later provide details of how this schema can be applied to each case.



Suppose that a node (region) **C** in a BSP has two children, say **D** and **E**. Then the boundary between **D** and **E** is a line segment, horizontal or vertical, that extends between two points on the boundary of **C**. This segment is called a *cut* of **C**.

Our proof is by induction. We use $\mathcal{N}$ to denote the given collection of the disjoint regions. For a subrectangle of **R**, say **C**, we define induced collection $\mathcal{N}_{\mathbf{C}}$, that consists of nonempty intersections of rectangles from $\mathcal{N}$ with **C**.

We start by declaring one of the sides of **R** to be *unbroken* and the other to be *broken*. Then we initially give money to the elements of $\mathcal{N} = \mathcal{N}_{\mathbf{R}}$ according to the following formula (for some x and y, with $x \geq y \geq 1$): a rectangle not adjacent to a broken side gets x dollars, a rectangle adjacent to only one broken side gets y dollars and any other rectangle gets 1 dollar. We will say that $\mathcal{N}_{\mathbf{R}}$ is *properly endowed*. The choice of x and y will vary for the two problems.

Our inductive step is the following. If a rectangle **C** has three broken sides and $\mathcal{N}_{\mathbf{C}}$ is properly endowed, then

- either $|\mathcal{N}_{\mathbf{C}}| = 1$,
- or, we can cut **C** into two rectangles **D** and **E**, declare three broken sides for these rectangles, and provide proper endowment for $\mathcal{N}_{\mathbf{D}}$ and $\mathcal{N}_{\mathbf{E}}$ by redistributing the money from the proper endowment of **A**.
- or, we can cut **C** twice, into **D**, **E** and **F**, and make the declarations of broken sides and redistribution of dollars for all three new rectangles (one child and two grandchildren of **C**).

This inductive argument is sufficient (with $x = 3$ for Theorem 1 and $x = 2$ for Theorem 2), because every leaf in the resulting BSP gets at least one dollar, and initially there are less then xn^* dollars; thus we have less than xn^* leaves in our BSP. Finally, it is easy to eliminate empty regions in the resulting BSP, if so desired, by ensuring that each region intersects at least one rectangle of our collection: if the region for a non-root node does not intersect any rectangle, then simply cover it be extending its sibling in the BSP, and if necessary, the descendants of this sibling.

The inductive step is easy if we can cut $\mathbf{C}$ without splitting any of the rectangles in $\mathcal{N}_\mathbf{C}$, i.e. we have $\mathcal{N}_\mathbf{C} = \mathcal{N}_\mathbf{D} \cup \mathcal{N}_\mathbf{E}$, $\mathcal{N}_\mathbf{C} \neq \mathcal{N}_\mathbf{D}$ and $\mathcal{N}_\mathbf{C} \neq \mathcal{N}_\mathbf{E}$. A side of the child rectangle that is a (part of a) broken side of $\mathbf{C}$ is declared broken. If a child has two undeclared sides, we we arbitrarily declare one of them to be broken. Since $x \geq y \geq 1$, it is easy to see that no rectangle from $\mathcal{N}_\mathbf{C}$ needs more money after this partition. We refer to this case as the *easy case* of the inductive step.

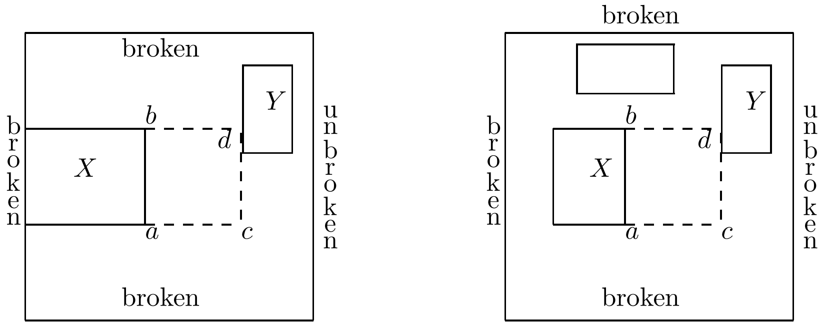
For ease of discussion, we assume that the unbroken side of $\mathbf{C}$ is the right side of $\mathbf{C}$ (by rotating the coordinate axes, if necessary).

2.2 Application of the General Schema

Now, we give details of the application of the general schema for each of the theorems.

2.2.1 Proof of Theorem 1

Set $x = 3$ and $y = 3/2$. Our algorithm is a modification of the algorithm discussed in [PY92]. Let X be the longest rectangle which is adjacent to the broken left side of $\mathbf{C}$, or, in the absence of any such rectangle, the rectangle in $\mathbf{C}$ whose left side is closest to the left side of $\mathbf{C}$ (break ties arbitrarily). Slide the right vertical side (a, b) of X until either it hits a rectangle in $\mathbf{C}$ or hits the unbroken side of $\mathbf{C}$. If (a, b) hits the unbroken side of $\mathbf{C}$, then this is the easy case of inductive step. Otherwise, let (c, d) be the position of (a, b) when it hits a rectangle Y of $\mathbf{C}$. Notice that if Y is adjacent to two broken (horizontal) sides of $\mathbf{C}$ then we have the easy case of the inductive case; hence we assume that this is not the case.



Our first cut is vertical through point d. If this cut does not split any rectangles of $\mathcal{N}_\mathbf{C}$, we have the easy case. Otherwise we will assume that this cut splits a rectangle above line (b, d). The sides of $\mathbf{D}$ $\mathbf{E}$ and $\mathbf{F}$ contributed by the first cut are declared broken. Our second cut will be the horizontal line (b, d) extended to the left side of $\mathbf{C}$. The sides of children rectangles that are adjacent to that line are declared unbroken. The second cut is the easy case of the inductive step, but was essential because it provided the left products of the cuts with unbroken sides. While checking if we have enough money for the endowments, we analyze the first cut only. It splits a number of rectangles in $\mathbf{C}$, among which at most two are adjacent to one broken side of $\mathbf{C}$ (one adjacent to the top side and the other to the bottom side of $\mathbf{C}$). Every other rectangle that is cut had 3 dollars before the cut, and after the cut, each of the two pieces need $3/2$ dollars, which can be obtained by distributing the 3 dollars evenly between them. Hence, there are two cases to consider:

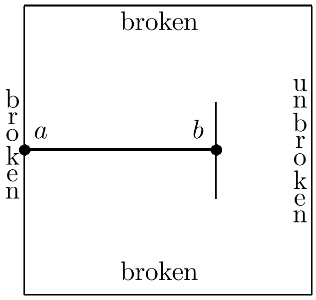
- Only one rectangle adjacent to a broken side of $\mathbf{C}$ is cut. Before cut, the rectangle had $3/2$ dollars. After the cut, each of the two pieces need one dollar; hence it needs an extra $1/2$ dollars for proper endowment. However, because the status of rectangle Y is changed, it has a surplus of at least $\min\{x - y, y - 1\} = 1/2$ dollars, which can be used for this purpose.

- Two rectangles adjacent to a broken side of **C** are cut. Then, each of these rectangles needs an additional of $1/2$ dollars. However, in this case, Y could not have been adjacent to any broken side of **C** before the cut but becomes adjacent to one broken side after the cut; hence it has a surplus of $x - y = 3/2$ dollars, which can be used to provide the additional one dollar.

The algorithm can be implemented using priority search trees and range queries much as in [PY92, dAF92]. Details are omitted. The total time taken is $O(n \log n)$.

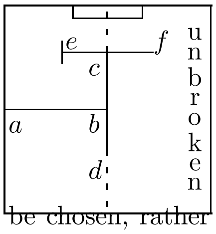
2.2.2 Proof of Theorem 2

Set $x = 2$ and $y = 1$. For a subrectangle $\mathbf{U}$ of $\mathbf{C}$ let $\mathcal{B}_{\mathbf{U}}$ to be the union of the boundary segments of the rectangles of $\mathcal{N}_{\mathbf{U}}$ that are not contained in the boundary of $\mathbf{U}$. We pick a horizontal line segment (a, b) with the following properties:



- a belongs to the left side of $\mathbf{C}$ (remember that the left vertical side of $\mathbf{C}$ is broken);
- (a, b) is a horizontal line segment;
- (a, b) is a subset of $\mathcal{B}_{\mathbf{C}}$;
- No point on (a, b), except possibly the points a and b, lie on a side of $\mathbf{C}$;
- (a, b) is a longest segment with the above properties.

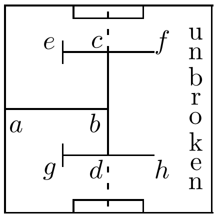
Note that b is not on a side of $\mathbf{C}$, otherwise we would have the easy case as described before. Our first cut is vertical through point b. The sides of **D** **E** and **F** contributed by this cut is declared broken. The second cut is (a, b) itself, and the sides of **D** **E** and **F** contributed by this cut is declared unbroken. As before, any side of **D** **E** and **F** that is part of a broken side of **C** is also declared broken. It remains to show that the endowment of $\mathcal{N}_{\mathbf{C}}$ is sufficient to provide the endowments of $\mathcal{N}_{\mathbf{D}}$, $\mathcal{N}_{\mathbf{E}}$ and $\mathcal{N}_{\mathbf{F}}$.



Point b must lie in the interior of a vertical segment from $\mathcal{B}_{\mathbf{C}}$ that is perpendicular to (a, b). We extend this segment maximally vertically in both directions, so it is a union of segments (c, b) and (b, d). If both c and d are on the boundary of $\mathbf{C}$, then cut (c, d) makes it an easy case. Thus we can assume that at least one of them, say c, is located in the interior of $\mathbf{C}$, and thus in the interior of some maximal horizontal segment from $\mathcal{B}_{\mathcal{B}}$, say (e, f). We assume that e is to the left of f.

Observe that e must be located in the interior of $\mathbf{C}$, since otherwise segment (e, f) would be chosen, rather than (a, b). Thus the rectangle of $\mathcal{N}_{\mathbf{C}}$ that is adjacent to segments (e, c) and (b, c) is not adjacent to the boundary of $\mathbf{C}$, and therefore it was endowed with two dollars. During our cuts this rectangle is not subdivided, and it is adjacent to our new broken side, thus we can take one dollar from its endowment to be used elsewhere. To perform the first cut, we extend (b, c) toward the boundary of $\mathbf{C}$. We may cut a number of rectangles from $\mathcal{N}_{\mathbf{C}}$. The last one is adjacent to a broken boundary segment of $\mathbf{C}$, so it has only one dollar, and after the cut we need one dollar for each of its two parts. This is where we used a dollar that we have extracted a moment ago. Other rectangles which are cut cannot be adjacent to a broken boundary, since the only possibility is if the rectangle is adjacent to the boundary segment where a is located, but then such a rectangle would provide a better (longer) choice for (a, b). Thus each of these rectangles has two dollars, but after the cut their parts are adjacent to the new broken boundary segment, so they need one dollar each.

Now we extend (b, d) in the opposite direction. If d is on the boundary of $\mathbf{C}$, then this extension does not cut any rectangles of $\mathcal{N}_{\mathbf{C}}$. Otherwise we cut some rectangles. By the same reasoning as before, we need money (one dollar) only for the last rectangle that is cut. It is easy to see that in this case point d is in the interior of a maximal segment from $\mathcal{B}_{\mathbf{C}}$, say (g, h). By the same reasoning as applied before to points c, e and f, we can conclude that the rectangle adjacent to to segments (b, d) and (g, d) is not adjacent to a broken side of $\mathbf{C}$, so it had two dollars, and becomes adjacent to a broken side after the partition, so it provides us with that additional one dollar.



The algorithm can be implemented in a manner similar to as described in [PY92, dAF92]. The only major additional step we need is to find the segment (a, b); this can be done using priority search trees and doing range queries. Hence, the total time taken by the algorithm is $O(n \log n)$.

3 Lower Bounds

In this section, we prove the following lower bounds:

Theorem 3 *There exists a configuration of n isothetic rectangles that form a tiling of the space such that any BSP tree for them is of size at least $3/2 n$. If the rectangles are not required to form a tiling, there exists a configuration of n rectangles such that any BSP tree for them is of size at least $2n$.*

Proof. We first focus on the tiling case.

We will construct the configuration in stages. See Figure 1. Stage 1 is the five figure configuration shown there. We reflect a Stage 1 configuration along each axes and the lower right corner to get a configuration of four copies of Stage 1. We merge any two mirror reflections we find abutting the borders of the four copies into one rectangle, whenever possible. The resulting configuration is shown in Figure 1 with 13 rectangles. That is Stage 2. Stage 3 comprises four configurations from Stage 2 merged along the borders as before, and so on.

Any BSP for Stage 1 needs to cut one of the rectangles and therefore generates at least one additional rectangle. We would like to claim that any BSP for Stage 2 needs at least 4 additional rectangles. This is not true as such for the 13 rectangle configuration shown in Figure 1. We need to make sure that the vertical line of one of the top two copies of Stage 1 does not align with the vertical line in the copy of Stage 1 directly underneath it. This is accomplished by moving the vertical lines marked in the figure a little along the direction of the arrow. Likewise for the horizontal lines of the right two copies of Stage 1. Now we can convince ourselves that at least four additional rectangles are needed in any BSP tree of Stage 2. Similarly, by moving various line segments a little along the direction shown in the figure, we can ensure that Stage 3 needs at least 16 additional rectangles, and so on.

That completes the description of the configuration. We will now complete the lower bound calculation. Define B_i to be the number of rectangles on any one side of Stage i configuration, R_i to be total number of rectangles in the Stage i configuration, and C_i to be the number of additional rectangles needed in any BSP tree for Stage i. Clearly, a lower bound *factor* on the BSP tree size is $1 + \frac{C_i}{R_i}$. Observe that $B_i = 2B_{i-1} - 1$ and $C_i = 4C_{i-1}$. Little reflection will reveal that $R_i = 4R_{i-1} - (4B_i - 1)$. As boundary conditions, we have $B_1 = 2$, $C_1 = 1$ and $R_1 = 5$. Solving the recurrences, we have $C_i = 4^{i-1}$ and $R_i = 2(4^{i-1}) + 2^i + 1$. Thus the lower bound on the BSP tree size is at least

$$1 + \frac{4^{i-1}}{2(4^{i-1}) + 2^i + 1}$$

which for i (hence n) large enough is roughly $3/2$.

For the case when the rectangles are not required to be a tiling, we repeat the same construction as above except that we leave the central square as a hole. The first two stages are shown in Figure 2 where the shaded region is a hole. Repeating the argument above, we obtain the lower bound to be at least

$$1 + \frac{4^{i-1}}{4^{i-1} + 2^i + 1}$$

which for i (hence n) large enough is roughly 2. That completes the proof. ❑

Remark. We can change the construction a little by stacking two copies of each stage to get a slightly improved lower bound, but since it does not change the main factors in the lower bound, we do not include that construction here.

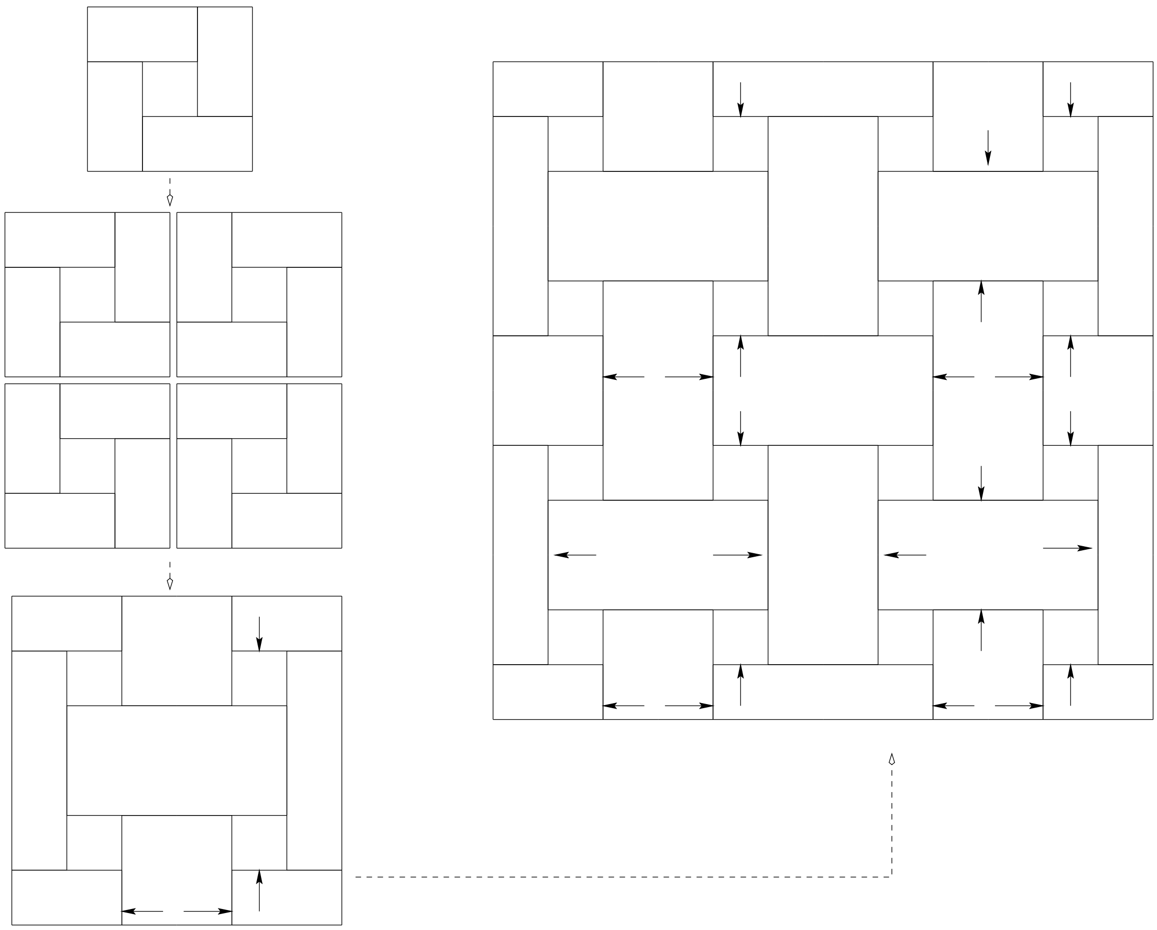


Figure 1: Lower bound construction for a set of isothetic rectangles that form a tiling. Solid arrows indicate the direction the side of a rectangle has to be moved slightly.

4 Application to Tiling Problems

Our upper bound on the BSP tree size of n isothetic rectangles that form a tiling leads to improved approximation algorithms for several tiling problems. As an example, consider the following [MPS99].

Dual Tiling Problem. Given an $n \times n$ array A of positive integers and a parameter $\delta > 0$, the problem is to tile A with the minimum number of axis-parallel, rectangular tiles such that the maximum weight of any tile is at most δ. The *weight* of a tile is the sum squared error of each entry from the average of the entries in that tile. $\blacksquare$

This is known to be NP-hard [MPS99]. Our results are:

- There is an $O(n^5)$ time algorithm to find a tiling with maximum weight of a tile being at most δ using no more than twice as many tiles as needed.

  The previously best known algorithm was a factor 2 approximation too, but it took $O(n^{\geq 10})$ time! [MPS99, KMP98]

- We can also obtain a variety of results on trading time for approximation for this problem. In particular, we obtain $O(n^{2+\epsilon})$ time algorithm that uses at most $2p^*O(1/\epsilon^2)$ tiles, where p^* is the minimum number of tiles needed.

  The previously best known algorithm with this running time was worse by a factor of 2 [MPS99].

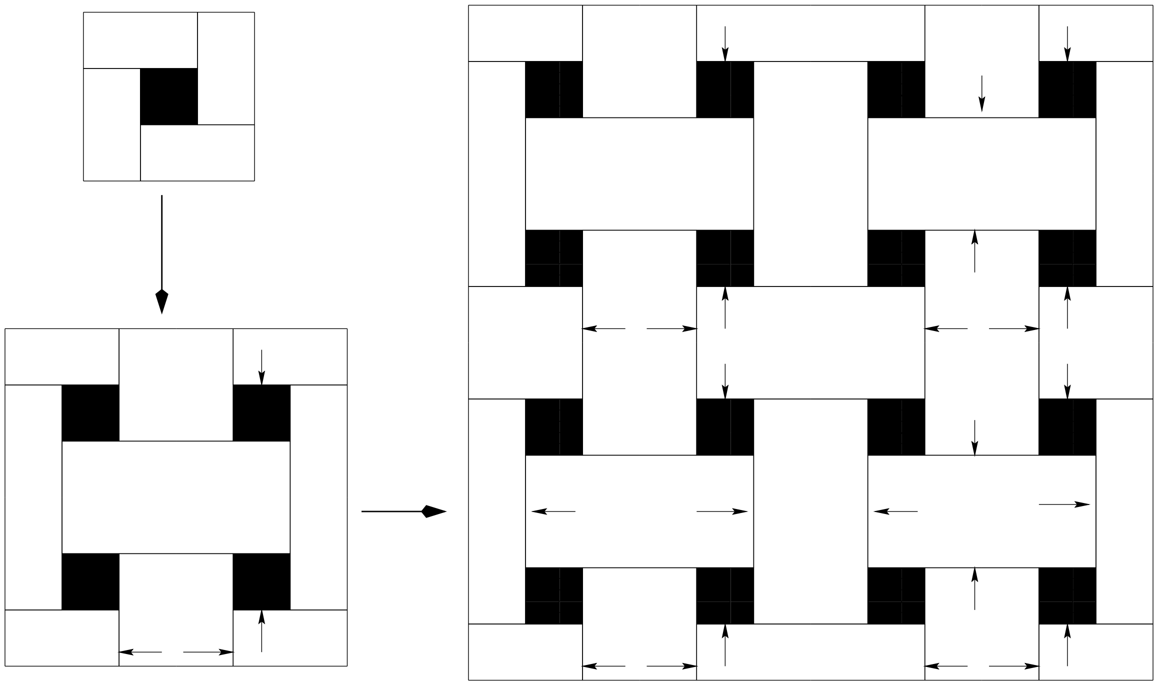


Figure 2: Lower bound construction for a set of isothetic rectangles, not necessarily a tiling. The black region is a hole.

Both the claims above depend on the following argument. Say the optimum tiling has p^* tiles. Our result implies that there is a BSP tree for those tiles of size at most $2p^*$. Since the weight of a subtile of a tile is no more than that of the tile itself, this means that there exists a BSP of the tiling with weight at most δ, but with at most $2p^*$ tiles. One can find a best BSP of the array A (with the tiles satisfying the weight bounds) in $O(n^5)$ time by dynamic programming, and this approximates p^* by a factor of 2. For the second approximation claim, we use the same reasoning, but trivially appeal to the approximations for the BSP inherent in [MPS99].

Improvements such as above hold for other tiling problems such as with either the maximum or the sum of weights of tiles, different weight functions etc.[2] The technical condition for our claims to hold is that the weight functions have to be superadditive. Informally, this means that splitting the tiles in any optimum solution does not make it worse in the criteria (formal definition can be found in [MPS99]). Since all these improvements follow from our upper bound for BSP tree size combined with the known techniques in [KMP98, MPS99], we do not discuss them any further here. We just note that these rectangular tiling problems (and improved approximation algorithms for them) are of interest in many areas: databases, parallel load balancing, spatial data structures, video compression etc. See [MPS99] for a detailed description.

5 Concluding Remarks

We have shown upper and lower bounds on the size of a BSP tree for a set of n isothetic rectangles. We have shown an upper bound of $3n$ in general and of $2n$ if the rectangles form a tiling; this improves the $12n$ bound in [PY92] and the $4n$ bound in [NW95, dAF92]. We present first nontrivial lower bounds of $2n$ for arbitrary isothetic rectangles and ${}^3/_2 n$ if the rectangles must form a tiling. The BSP tree is one of the most popular

[2]This claims also applies to the special case when the weight of a tile is the sum of all items in it. This is the DRTILE problem considered in [KMP98] and elsewhere. Many results exist for the DRTILE problem which use the special properties of the weight function and obtain better bounds than the ones we have quoted above. But the strength of our results here is that they apply to many more complex weight functions, sum-squared-error being one example.

data structures and the performance of the applications that rely on it depend on its size. Even "small" factor improvements such as $4/3$ or 2 we show here on the size of BSP trees will be of importance. In addition, our results give improved approximation algorithms for rectangular tiling problems that arise in many application areas [MPS99]. We leave open the problem of closing the gap between the upper and lower bounds, and of developing such detailed analyses for other objects and their associated BSP trees.

References

[AG+96] P. Agarwal, E. Grove, T. Murali and J. Vitter. Binary space partitions for fat rectangles. *Proc. Symp. Found. of Comp. Sci. (FOCS)*, 1996.

[AS94] P. Agarwal and S. Suri. Surface approximation and geometric partitions. *Proc. 5th ACM-SIAM Discrete Algorithms*, 24-33, 1994.

[B93] C. Ballieux. Motion planning using binary space partitions. *Technical Report*, Utrecht University, 1993.

[CF89] S. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *Computer graphics*, 23:99-106, 1989.

[dAF92] F. d'Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Proc. Letters*, 44, 255–259, 1992.

[deGO] Mark de Berg, Marko de Groot and Mark Overmars. New results on binary space partitions in the plane. *Proc. 4th Scand. Workshop Algo. Theory*, Vol 824, LNCS, 61-72, 1994.

[D94] S. E. Dorward. A survey of object-space hidden surface removal. *Intern. J. Comput. Geom. Appl*, 4, 325-362, 1994.

[FKN80] H. Fuchs, Z. Kedem and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3), 124-133, 1980.

[NW95] V. Hai Nguyen and P. Widmayer. Binary space partitions for sets of hyperrectangles. LNCS, 1023, 1995.

[KMP98] S. Khanna, S. Muthukrishnan, and M. Paterson. Approximating rectangle tiling and packing. *Proc Symp. on Discrete Algorithms (SODA)*, 384-393, 1998.

[M90] J. Mitchell. On maximum flows in polyhedral domains, *J. Comput. Syst. Sci.*, 40, 88-123, 1990.

[MPS99] S. Muthukrishnan, V. Poosala and T. Suel. Rectangular Partitionings: Algorithms, Complexity and Applications. *Proc. Intl. Conf. on Database Theory*, 1999.

[NT86] B. Naylor and W. Thibault. Application of BSP trees to ray-tracing and CSG evaluation. Technical Report GIT-ICS 86/03, Georgia Tech., Feb 1986.

[PY90] M. Paterson and F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5, 485-503, 1990.

[PY92] M. Paterson and F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13, 99-113, 1992.

[S90] H. Samet. *Applications of spatial data structures: computer graphics, image processing and GIS*, Addison-Wesley, 1990.

[T92] S. Teller. Visibility computations in densely occluded polyhedral environments. *PhD Thesis*, Dept of Computer Science, University California, Berkeley, 1992.

[TN87] W. Thibault and B. Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics (SIGGRAPH)*, 21, 153-162, 1987.

[van90] P. van Oosterom. A modified binary space partition for geographic information systems. *Int. J. GIS*, 4(2), 133-146, 1990.