Finding perfect auto-partitions is NP-hard

Finding Perfect Auto-Partitions is NP-hard*

Mark de Berg† Amirali Khosravi†

Abstract

A perfect BSP for a set S of disjoint line segments in the plane is a BSP in which none of the objects is cut. We study a specific class of BSPs, called *auto-partitions* and we prove that it is NP-hard to find if a perfect auto-partition exists for a set of lines.

1 Introduction

Many problems involving objects in the plane or higher-dimensional space are solved more efficiently if a hierarchical partitioning of the space is given. One of the most popular hierarchical partitioning schemes is the *binary space partition*, or BSP for short. In a BSP the space is recursively partitioned by hyperplanes until there is at most one object intersecting the interior of each cell in the final partitioning. Note that the splitting hyperplanes not only partition the space, they may also cut the objects into fragments.

The recursive partitioning can be modeled by a tree structure, called a BSP *tree.* Nodes in a BSP tree corresponds to subspaces of the original space, with the root node corresponding to the whole space and the leaves corresponding to the cells in the final partitioning. Each internal node stores the hyperplane used to split the corresponding subspace, and each leaf stores the object fragment intersecting the corresponding cell.

BSPs have been used in numerous applications. In most of these applications, the efficiency is determined by the *size* of the BSP tree, which is equivalent to the total number of object fragments created by the partitioning process. As a result, many algorithms have been developed that create small BSPs. For example, Paterson and Yao [6] presented an algorithm that computes for any given set of n disjoint segments in the plane a BSP of size $O(n \log n)$. Also for many other settings—axis-parallel objects, 3-dimensional objects, fat objects, etc.—algorithms have been developed that produce provably small BSPs; for an overview of results and applications on BSPs see the survey paper by Tóth [7].

In all of the above algorithms, bounds are proved on the *worst-case size* of the computed BSP *over all sets of n input objects* from the class of objects being considered. For any particular input, one may be able to do much better than in a worst-case scenario. Ideally, one would like to have an algorithm that computes a BSP that is *optimal for the given input*, rather than optimal in the worst-case. For n axis-parallel segments in the plane one can compute an optimal rectilinear BSP in $O(n^5)$ time [2]. Another result related to optimal BSPs is that for any set of (not necessarily rectilinear) disjoint segments in the plane one can compute a *perfect* BSP in $O(n^2)$ time, if it exists [1]. (A perfect BSP is a BSP in which none of the objects is cut). If such a BSP does not exist, then the algorithm only reports this fact.

Thus, it is still unknown if it is possible to compute (or maybe approximate) an optimal BSP for a set of segments. We study this problem for a specific type of BSPs, called *auto-partitions.* Let S denote the set of n disjoint input segments, R the region which we want to partition at some point, and $S(R)$ the set of segment fragments in the interior of R. Then an auto-partition uses a splitting line that contains a segment from $S(R)$. Fig. 1 shows a general (unrestricted) BSP of a set of input segments and an auto-partition of it.

Since autopartitions are a restricted type of BSPs, our hope was that it would be easier to compute optimal autopartitions than it is to compute optimal unrestricted BSPs. Unfortunately, this turns out to be not the case: we show that even the problem of finding perfect autopartitions—that is, deciding if the minimum number of cuts is zero—is already NP-hard. This should be contrasted to the result mentioned above, that deciding whether a set of segments admits a perfect general (or unrestricted) BSP can be done in $O(n^2)$ time. Hence, optimal auto-partitions seem more difficult to compute than optimal unrestricted BSPs.

2 Hardness of computing perfect auto-partitions

We consider the following problem.

PERFECT AUTO-PARTITION
Input: A set S of n disjoint line segments in the plane.
Output: YES if S admits a perfect auto-partition(an auto-partition without cuts), NO otherwise.

We will show that PERFECT AUTO-PARTITION is

*This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

†Department of Computing Science, TU Eindhoven, mdberg@win.tue.nl, akhosrav@win.tue.nl

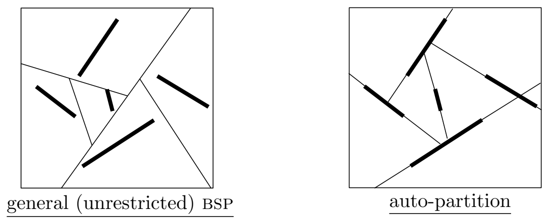

Figure 1: Two types of BSPs. Note that, as is usually done for auto-partitions, we have continued the auto-partition until the cells are empty.

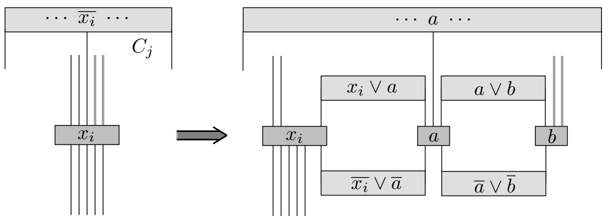
$\cdots\ \overline{x_i}\ \cdots$
C_j
x_i
$\cdots\ a\ \cdots$
$x_i \vee a$
$a \vee b$
x_i
a
b
$\overline{x_i} \vee \overline{a}$
$\overline{a} \vee \overline{b}$

Figure 2: replacing an inconsistent variable-clause.

NP-hard. Our proof is by reduction from a special version of the satisfiability problem, which we define and prove NP-complete in the next subsection. After that we prove the hardness of Perfect Auto-Partition.

2.1 Planar monotone 3-SAT

Let $\mathcal{U} := \{x_1, \ldots, x_n\}$ be a set of n boolean variables, and let $\mathcal{C} := C_1 \wedge \cdots \wedge C_m$ be a CNF formula defined over these variables, where each clause C_i is the disjunction of at most three variables. 3-SAT is the problem of deciding whether such a boolean formula is satisfiable. An instance of 3-SAT is called *monotone* if each clause consists only of positive variables or only of negative variables. 3-SAT is NP-complete, even when restricted to monotone instances [3].

For a given (not necessarily monotone) 3-SAT instance, consider the bipartite graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{C}, \mathcal{E})$, where there is an edge $(x_i, C_j) \in \mathcal{E}$ if and only if x_i or its negation $\overline{x_i}$ is one of the variables in the clause C_j. Liechtenstein [5] has shown that 3-SAT remains NP-complete when $\mathcal{G}$ is planar. Moreover, as shown by Knuth and Raghunatan [4], one can always draw the graph $\mathcal{G}$ of a planar 3-SAT instance such that the variables and clauses are drawn as rectangles with all the variable-rectangles on a horizontal line, the edges connecting the variables to the clauses are vertical segments, and the drawing is crossing-free. We call such a drawing of a planar 3-SAT instance a *rectilinear representation.* Planar 3-SAT remains NP-complete when a rectilinear representation is given.

Next we introduce a new version of 3-SAT, which combines the properties of monotone and planar instances. We call a clause with only positive variables a *positive clause*, a clause with only negative variables a *negative clause*, and a clause with both positive and negative variables a *mixed clause.* Thus a monotone 3-SAT instance does not have mixed clauses. Now consider a 3-SAT instance that is both planar and monotone. A *monotone rectilinear representation* of such an instance is a rectilinear representation where all positive clauses are drawn on the positive side of (that is, above) the variables and all negative clauses are drawn on the negative side of (that is, below) the variables. Our 3-SAT variant is defined as follows.

Planar Monotone 3-sat
Input: A monotone rectilinear representation of a planar monotone 3-SAT instance.
Output: YES if it is satisfiable, NO otherwise.

Planar Monotone 3-sat is obviously in NP. We will prove that Planar Monotone 3-sat is NP-hard by a reduction from Planar 3-sat.

Let $\mathcal{C} = C_1 \wedge \cdots \wedge C_m$ be a given rectilinear representation of a planar 3-SAT instance defined over the variable set $\mathcal{U} = \{x_1, \ldots, x_n\}$. We call a variable-clause pair *inconsistent* if the variable is negative in that clause while the clause is placed on the positive side of the variables, or the variable is positive in the clause while the clause is placed on the negative side. If a rectilinear representation does not have inconsistent variable-clause pairs, then it must be monotone. Indeed, any monotone clause must be placed on the correct side of the variables, and there cannot be any mixed clauses because any mixed clause must form an inconsistent pair with at least one of its variables. We convert the given instance $\mathcal{C}$ step by step into an equivalent instance with a monotone planar representation, in each step reducing the number of inconsistent variable-clause pairs by one.

Let $(\overline{x_i}, C_j)$ be an inconsistent pair; inconsistent pairs involving a positive variable in a clause on the negative side can be handled similarly. We get rid of this inconsistent pair as follows. We introduce two new variables, a and b, and modify the set of clauses.

- In clause C_j, replace $\overline{x_i}$ by a.
- Introduce the following four clauses: $(x_i \vee a) \wedge (\overline{x_i} \vee \overline{a}) \wedge (a \vee b) \wedge (\overline{a} \vee \overline{b})$.
- In each clause containing x_i that is placed on the positive side of the variables and that connects to x_i to the right of C_j, replace x_i by b.

Let $\mathcal{C}'$ be the new set of clauses. The proof of the following lemma is omitted in this abstract.

Lemma 1 *$\mathcal{C}$ is satisfiable iff $\mathcal{C}'$ is satisfiable.*

Fig. 2 shows how this modification is reflected in the rectilinear representation.

By applying the above conversion to each of the at most $3m$ inconsistent variable-clause pairs, we obtain a 3-SAT instance with at most $13m$ clauses defined over at most $n + 6m$ variables. This new instance is satisfiable iff $\mathcal{C}$ is satisfiable, and it has a monotone representation. We get the following theorem.

Theorem 2 PLANAR MONOTONE 3-SAT *is* NP-*complete.*

2.2 From planar monotone 3-SAT to perfect auto-partitions

Let $\mathcal{C} = C_1 \wedge \cdots \wedge C_m$ be a planar monotone 3-SAT instance defined over a set $\mathcal{U} = \{x_1, \ldots, x_n\}$ of variables, with a monotone rectilinear representation. We will show how to construct a set S of line segments that admits a perfect auto-partition iff $\mathcal{C}$ is satisfiable. The idea behind the reduction is shown in Fig. 3.

The variable gadget. For each variable x_i there is a gadget consisting of two segments, s_i and $\overline{s_i}$. Setting $x_i =$ TRUE corresponds to extending s_i before $\overline{s_i}$, and setting $x_i =$ FALSE to extending $\overline{s_i}$ before s_i.

The clause gadget. For each clause C_j there is a gadget consisting of four segments, $t_{j,0}, \ldots, t_{j,3}$. The segments in a clause form a cycle, that is, the splitting line $\ell(t_{j,k})$ cuts the segment $t_{j,(k+1) \bmod 4}$. This means that a clause gadget in isolation, would generate at least one cut. Now suppose that the gadget for C_j is crossed by the splitting line $\ell(s_i)$ through the segment s_i in such a way that $\ell(s_i)$ separates the segments $t_{j,0}, t_{j,3}$ from $t_{j,1}, t_{j,2}$, as in Fig. 3. Then the cycle is broken by $\ell(s_i)$ and no cut is needed. This does not work when $\ell(\overline{s_i})$ is used before $\ell(s_i)$, since then $\ell(s_i)$ is blocked by $\ell(\overline{s_i})$ before crossing C_j.

The idea is as follows. For each clause $(x_i \vee x_j \vee x_k)$, we want to make the splitting lines $\ell(s_i)$, $\ell(s_j)$, and $\ell(s_k)$ all cross the clause gadget. Then by setting one of these variables to TRUE, the cycle is broken and no cuts are needed for the clause. We must be careful that the splitting lines are not blocked in the wrong way—for example, it could be problematic if $\ell(\overline{s_k})$ would block $\ell(s_i)$—and also that clause gadgets are only intersected by the splitting lines corresponding to the variables in that clause. In the remainder of this section we show how to overcome these problems.

Detailed construction. From now on we assume that the variables are numbered based on the monotone rectilinear representation, with x_1 being the leftmost and x_n being the rightmost variable.

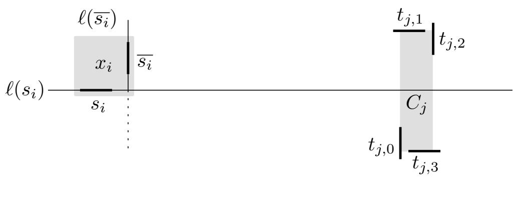

Figure 3: The idea behind the reduction.

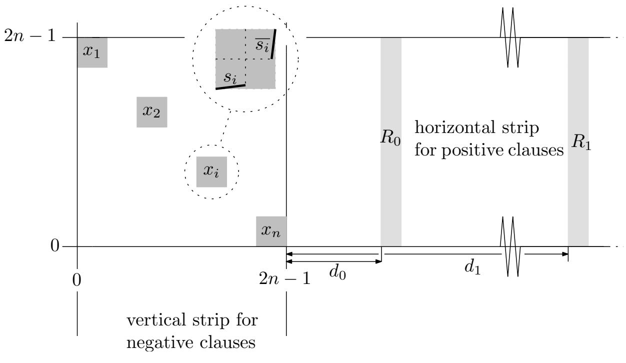

Figure 4: Placement of the variable gadgets and the clause gadgets (not to scale).

The gadget for a variable x_i will be placed inside the unit square $[2i-2 : 2i-1] \times [2n-2i : 2n-2i+1]$, as illustrated in Fig. 4. The segment s_i is placed with one endpoint at $(2i-2, 2n-2i)$ and the other endpoint at $(2i-\frac{3}{2}, 2n-2i+\varepsilon_i)$ for some $0 < \varepsilon_i < \frac{1}{4}$. The segment $\overline{s_i}$ is placed with one endpoint at $(2i-1, 2n-2i+1)$ and the other endpoint at $(2i-1-\overline{\varepsilon_i}, 2n-2i+\frac{1}{2})$ for some $0 < \overline{\varepsilon_i} < \frac{1}{4}$. Next we specify the slopes of the segments, which determine the values ε_i and $\overline{\varepsilon_i}$.

The gadgets for the positive clauses will be placed to the right of the variables, in the horizontal strip $[-\infty : \infty] \times [0 : 2n-1]$; the gadgets for the negative clauses will be placed below the variables, in the vertical strip $[0 : 2n-1] \times [-\infty : \infty]$. We describe how to choose the slopes of the segments s_i and how to place the positive clauses; the segments $\overline{s_i}$ and the negative clauses are handled in a similar fashion.

Consider the set $\mathcal{C}^+$ of all positive clauses in our 3-SAT instance, and the way they are placed in the monotone rectilinear representation. We call the clause directly enclosing a clause C_j the *parent* of C_j. Now let $\mathcal{G}^+ = (\mathcal{C}^+, \mathcal{E}^+)$ be the directed acyclic graph where each clause C_j has an edge to its parent (if it exists), and consider a topological order on the nodes of $\mathcal{G}^+$. We define the *rank* of a clause C_j, denoted by $\text{rank}(C_j)$, to be its rank in this topological order. If $\text{rank}(C_j) = k$ then C_j is placed in a $1 \times (2n+1)$ rectangle R_k at distance d_k from the line $x = 2n-1$ (see Fig. 4), where $d_k := 2 \cdot (2n)^{k+1}$.

Before describing how the clause gadgets are placed inside these rectangles, we define the slopes of the segments s_i. Define $\text{rank}(x_i)$, the rank of a variable x_i (with respect to the positive clauses), as the maximum rank of any clause it participates in. Now the slope of s_i is $\frac{1}{2 \cdot d_k}$, where $k = \text{rank}(x_i)$. Recall that x_i is placed inside the unit square $[2i-2 : 2i-1] \times [2n-2i : 2n-2i+1]$. The proof of the following lemma is omitted in this version.

Lemma 3 *Let x_i be a variable, and $\ell(s_i)$ be the splitting line containing s_i.*

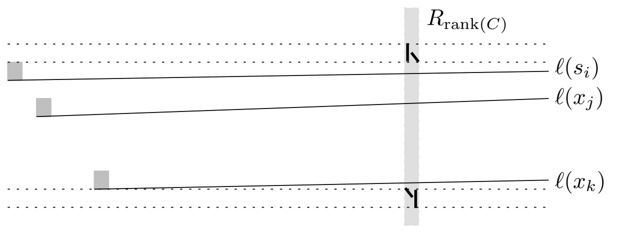

Figure 5: Placement of the segments forming a clause.

(i) For all x-coordinates in the interval $[2i-2 : 2n-1+d_{\text{rank}(x_i)}+1]$, $\ell(s_i)$ *has a y-coordinate in the range* $[2n-2i : 2n-2i+1]$.
(ii) $\ell(s_i)$ *intersects rectangles* $R_k (0 \le k \le \text{rank}(x_i))$.
(iii) $\ell(s_i)$ *does not intersect any rectangle* $R_k (k > \text{rank}(x_i))$.

We can now place the clause gadgets. Consider a clause $C = (x_i \vee x_j \vee x_k) \in \mathcal{C}^+$, with $i < j < k$; the case where C contains only two variables is similar. By Lemma 3(ii), the splitting lines $\ell(x_i), \ell(x_j), \ell(x_k)$ all intersect the rectangle $R_{\text{rank}(C)}$. Moreover, by Lemma 3(i) and since we have placed the variable gadgets one unit apart, there is a 1×1 square in $R_{\text{rank}(C)}$ just above $\ell(s_i)$ that is not intersected by any splitting line. Similarly, just below $\ell(s_k)$ there is a square that is not crossed. Hence, if we place the segments forming the clause gadget as in Fig. 5, then the segments will not be intersected by any splitting line. Moreover, the splitting lines of segments in the clause gadget—these segments either have slope -1 or are vertical—will not intersect any other clause gadget. This finishes the construction. The next lemma, whose proof is omitted, states the two key properties of construction. We say that, a splitting line $\ell(s_i)$ is *blocked* by $\ell(s_j)$ if $\ell(s_j)$ intersects $\ell(s_i)$ before $\ell(s_i)$ reaches $R_{\text{rank}(x_i)}$. This may prevent us from using $\ell(s_i)$ to resolve the cycle in the gadget of a clause containing x_i and is dangerous.

Lemma 4 *The variable and clause gadgets are placed such that the following holds:*
(i) The gadget for clause $(x_i \vee x_j \vee x_k)$ *is only intersected by the splitting lines* $\ell(s_i)$, $\ell(s_j)$ *and* $\ell(s_k)$. *Similarly, the gadget for clause* $(\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$ *is only intersected by the splitting lines* $\ell(\overline{s_i})$, $\ell(\overline{s_j})$ *and* $\ell(\overline{s_k})$.
(ii) A splitting line $\ell(s_i)$ *can only be blocked by a splitting line* $\ell(s_j)$ *or* $\ell(\overline{s_j})$ *when* $j \ge i$*; the same holds for* $\ell(\overline{s_i})$.

Theorem 5 Perfect Auto-Partition *is* np-*complete.*

Proof. We can verify in polynomial time whether a given ordering of applying the splitting lines yields a perfect auto-partition, so Perfect Auto-Partition is in np.

To prove that Perfect Auto-Partition is np-hard, take a Planar Monotone 3-sat instance and apply the above reduction to obtain a set S of $2n+4m$ segments forming an instance of Perfect Auto-Partition. The reduction can be done such that the segments have endpoints with integer coordinates of size $O(n^{2m})$, which means the number of bits for describing the instance is polynomial in $n+m$. It remains to show that $\mathcal{C}$ is satisfiable iff S has a perfect auto-partition.

Suppose S has a perfect auto-partition. Set $x_i :=$ true if s_i is extended before $\overline{s_i}$, and $x_i :=$ false otherwise. Since the auto-partition is perfect, the cycle in the gadget for a clause C must be broken. By Lemma 4(i) this can only be done by a line of one of the variables in the clause, say x_i. But then s_i has been extended before $\overline{s_i}(x_i = \text{true})$ and C is true.

Now consider a truth assignment to the variables that satisfies $\mathcal{C}$. A perfect auto-partition for S can be obtained as follows. When $x_1 =$ true we first take the splitting line $\ell(s_1)$ and then the splitting line $\ell(\overline{s_1})$; if $x_i =$ false then we first take $\ell(\overline{s_1})$ and then $\ell(s_i)$. Next we treat s_2 and $\overline{s_2}$ in a similar way, and so on. So far we have not made any cuts. We claim that after having put all splitting lines $\ell(s_i)$ and $\ell(\overline{s_i})$ in this manner, we can put the splitting lines containing the segments in the clause gadgets, without making any cuts. Indeed, consider the gadget for a positive clause C. Because the truth assignment is satisfying, one of its variables, x_i, is true. Then $\ell(s_i)$ is used before $\ell(\overline{s_i})$. Moreover, because we treated the segments in order, $\ell(s_i)$ is used before any other splitting lines $\ell(s_j)$, $\ell(\overline{s_j})$ with $j > i$ are used. By Lemma 4(ii) these are the only splitting lines that could block $\ell(s_i)$. Hence, $\ell(s_i)$ reaches the gadget for C and so we can resolve the cycle and get a perfect auto-partition. $\square$

References

[1] M. de Berg, M.M. de Groot, M.H. Overmars. Perfect binary space partitions. *Comput. Geom. Theory Appl.*,1997.

[2] M. de Berg, E. Mumford, B. Speckmann. Optimal BSPs and rectilinear cartograms. In *Proc. 14th Int. Symp. Advances Geographic Inf. Syst.*,2006.

[3] M.R. Garey, D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co.,1979.

[4] D.E. Knuth, A. Raghunathan. The problem of compatible representatives. *Discr. Comput. Math.*,1992.

[5] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*,1982.

[6] M.S. Paterson, F.F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discr. Comput. Geom.*,1990.

[7] C.D. Tóth, Binary space partitions: recent developments. *Combinat. and Comput. Geom.*, 2005.