Optimal Binary Space Partitions in the Plane*

Mark de Berg and Amirali Khosravi

TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands

Abstract. An optimal BSP for a set S of disjoint line segments in the plane is a BSP for S that produces the minimum number of cuts. We study optimal BSPs for three classes of BSPs, which differ in the splitting lines that can be used when partitioning a set of fragments in the recursive partitioning process: *free* BSPs can use any splitting line, *restricted* BSPs can only use splitting lines through pairs of fragment endpoints, and *auto-partitions* can only use splitting lines containing a fragment. We obtain the two following results:

- It is NP-hard to decide whether a given set of segments admits an auto-partition that does not make any cuts.
- An optimal restricted BSP makes at most 2 times as many cuts as an optimal free BSP for the same set of segments.

1 Introduction

Motivation. Many problems involving objects in the plane or some higher-dimensional space can be 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 [1]. 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 correspond 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.[1]

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 equal to the total number of object fragments created by the partitioning process. As a result,

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

[1] When the objects are $(d-1)$-dimensional—for example, a BSP for line segments in the plane—then it is sometimes required that the cells do not have any object in their interior. In other words, each fragment must end up being contained in a splitting plane. The fragments are then stored with the splitting hyperplanes containing them, rather than at the leaves. In particular, this is the case for so-called auto-partitions.

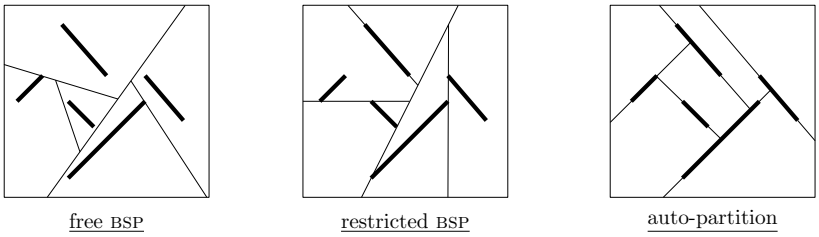

Fig. 1. The three types of BSPs, drawn inside a bounding box of the scene. Note that, as is usually done for auto-partitions, we have continued the auto-partition until the cells are empty.

many algorithms have been developed that create small BSPs; see the survey paper by Tóth [8] for an overview. In all these 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. 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. In other words, given an input set S, one would like to compute a BSP that is optimal (that is, makes the minimum number of cuts) for S.

For axis-aligned segments in the plane, one can compute an optimal rectilinear BSP for n axis-parallel segments in $O(n^5)$ time using dynamic programming [3]. Another result related to optimal BSPs is that for any set of (not necessarily rectilinear) disjoint segments in the plane one can compute a so-called *perfect* BSP in $O(n^2)$ time, if it exists [2]. (A perfect BSP is a BSP in which none of the objects is cut.) If a perfect BSP does not exist, then the algorithm only reports this fact; it does not produce any BSP in this case. Thus for arbitrary sets of segments in the plane it is unknown whether one can efficiently compute an optimal BSP.

Problem statement and our results. In our search for optimal BSPs, we consider three types of BSPs. These types differ in the splitting lines they are allowed to use. Let S denote the set of n disjoint segments for which we want to compute a BSP, and suppose at some point in the recursive partitioning process we have to partition a region R. Let $S(R)$ be the set of segment fragments lying in the interior of R. Then the three types of BSPs can use the following splitting lines to partition R.

- *Free* BSP*s* can use any splitting line.
- *Restricted* BSP*s* must use a splitting line containing (at least) two endpoints of fragments in $S(R)$. We call such a splitting line a *restricted splitting line*.
- *Auto-partitions* must use a splitting line that contains a segment from $S(R)$.

Fig. 1 illustrates the three types of BSPs. Note that an auto-partition is only allowed to use splitting lines containing a fragment lying in the region to be split; it is not allowed to use a splitting line that contains a fragment lying in a different region. Also note that when a splitting line contains a fragment—such

splitting lines must be used by auto-partitions, but may be used by the other types of BSPs as well—then that fragment is no longer considered in the rest of the recursive partitioning process. Hence, it will not be fragmented further.

We use $\text{OPT}_{\text{free}}(S)$ to denote the minimum number of cuts in any free BSP for S. Thus the number of fragments in an optimal free BSP for S is $n+\text{OPT}_{\text{free}}(S)$. Similarly, we use $\text{OPT}_{\text{res}}(S)$ and $\text{OPT}_{\text{auto}}(S)$ to denote the minimum number of cuts in any restricted BSP and in any auto-partition for S, respectively. Clearly, $\text{OPT}_{\text{free}}(S) \leqslant \text{OPT}_{\text{res}}(S) \leqslant \text{OPT}_{\text{auto}}(S)$. It is well known that for some sets of segments $\text{OPT}_{\text{res}}(S) < \text{OPT}_{\text{auto}}(S)$; indeed, it is easy to come up with an example where $\text{OPT}_{\text{res}}(S) = 0$ and $\text{OPT}_{\text{auto}}(S) = n/3$. Nevertheless, auto-partitions seem to perform well in many situations. Moreover, the collection of splitting lines to choose from in an auto-partition is smaller than for restricted or free BSPs, and so computing optimal auto-partitions might be easier than computing optimal restricted or free BSPs. Unfortunately, our hope to find an efficient algorithm for computing optimal auto-partitions turned out to be idle: in Section 2 we prove that computing optimal auto-partitions is an NP-hard problem. In fact, it is even NP-hard to decide whether a set of segments admits a perfect auto-partition. This should be contrasted to the result mentioned above, that deciding whether a set of segments admits a perfect restricted BSP can be done in $O(n^2)$ time. (Notice that when it comes to perfect BSPs, there is no difference between restricted and free BSPs: if there is a perfect free BSP then there is also a perfect restricted BSP [2].) Hence, optimal auto-partitions seem more difficult to compute than optimal restricted or free BSPs. Our hardness proof is based on a new 3-SAT variant, *monotone planar* 3-SAT , which we define and prove NP-complete in Section 2. We believe this new 3-SAT variant is interesting in its own right, and may find applications in other NP-completeness proofs. Indeed, our 3-SAT variant has already been used in a recent paper [9] to prove the NP-hardness of a problem on so-called switch graphs.

We turn our attention in Section 3 to unrestricted and free BSPs. In particular, we study the relation between optimal free BSPs and optimal restricted BSPs. In general, free BSPs are more powerful than restricted BSPs: in his MSc thesis[4], Clairbois gave an example of a set of segments for which the optimal free BSP makes one cut while the optimal restricted BSP makes two cuts, and he also proved that $\text{OPT}_{\text{res}}(S) \leqslant 3 \cdot \text{OPT}_{\text{free}}(S)$ for any set S. In Section 3 we improve this result by showing that $\text{OPT}_{\text{res}}(S) \leqslant 2 \cdot \text{OPT}_{\text{free}}(S)$ for any set S.

2 Hardness of computing perfect auto-partitions

Recall that an auto-partition of a set S of disjoint line segments in the plane is a BSP in which, whenever a subspace is partitioned, the splitting line contains one of the fragments lying in that subspace. We call an auto-partition *perfect* if none of the input segments is cut, and 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, NO otherwise.

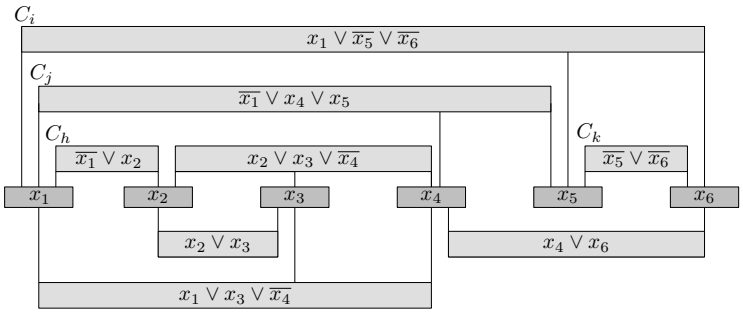

Fig. 2. A rectilinear representation of a planar 3-SAT instance.

We will show that PERFECT AUTO-PARTITION is 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.

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. Then 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 is monotone, that is, each clause consists only of positive variables or only of negative variables. 3-SAT is NP-complete, even when restricted to monotone instances [5].

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. Lichtenstein [7] has shown that 3-SAT remains NP-complete when $\mathcal{G}$ is planar. Moreover, as shown by Knuth and Raghunatan [6], one can always draw the graph $\mathcal{G}$ of a planar 3-SAT instance as in Fig. 2: 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 the instance is satisfiable, no otherwise.

Planar Monotone 3-sat is obviously in np. We will prove that it 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 as follows.

- 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 next lemma is in the appendix.

Lemma 1. *$\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is satisfiable.*

Fig. 3 shows how this modification is reflected in the rectilinear representation. (In this example, there are two clauses for which x_i is replaced by b, namely the ones whose edges to x_i are drawn fat and in grey.) We keep the rectangle for x_i at the same location. Then we shift the vertical edges that now connect to b instead of x_i a bit to the right—because of this, we may have to slightly grow or shrink some of the clause rectangles as well—to make room for a and b and the four new clauses. This way we keep a valid rectilinear representation.

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

Theorem 1. Planar Monotone 3-sat *is* np*-complete.*

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 $\mathcal{U} = \{x_1, \ldots, x_n\}$

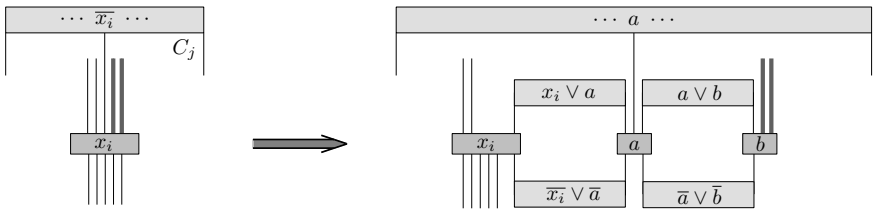

Fig. 3. Getting rid of an inconsistent variable-clause pair.

of variables, with a monotone rectilinear representation. We show how to construct a set S of line segments in the plane that admits a perfect auto-partition if and only if $\mathcal{C}$ is satisfiable. The idea is illustrated in Fig. 4.

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 corresponds 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, when considered in isolation, would generate at least one cut. Now suppose that the gadget for C_j is crossed by the splitting line $\ell(s_i)$ 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. 4. Then the cycle is broken by $\ell(s_i)$ and no cut is needed for the clause. But 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 thus as follows. For each clause $(x_i \vee x_j \vee x_k)$, we want to make sure that 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 to create a perfect autopartition for the segments in the clause. We must be careful, though, 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. Next we show how to overcome these problems.

Fig. 4. The idea behind replacing clauses and variables (C_j contains variable $\overline{x_i}$).

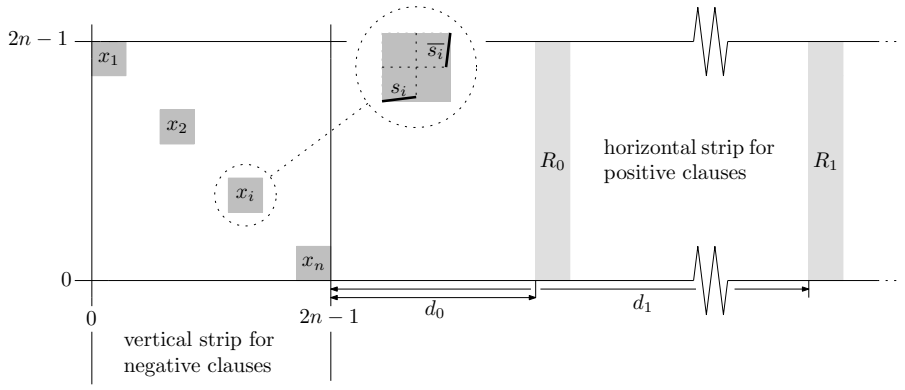

Fig. 5. Placement of the variable gadgets and the clause gadgets (not to scale).

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

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. 5. 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 $\overline{\varepsilon_i}$, and the placement of the clause gadgets.

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 to place the positive clauses; the segments $\overline{s_i}$ and the negative clauses are handled in a similar way.

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. In Fig. 2 for example C_i is the parent of C_j and C_k but it is not the parent of C_h. 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. Clause C_j will be placed at certain distance from the variables that depends on its rank. More precisely, 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. 5), 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

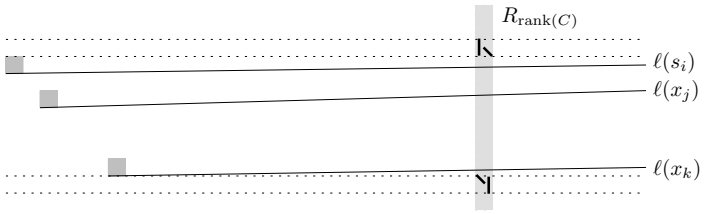

Fig. 6. Placement of the segments forming a clause gadget.

is placed inside the unit square $[2i-2, 2i-1] \times [2n-2i, 2n-2i+1]$. The proof of the following lemma is given in the appendix.

Lemma 2. *Let x_i be a variable, and $\ell(s_i)$ be the splitting line containing s_i.*
(i) For all x-coordinates in the interval $[2i-2, 2n-1+d_{\text{rank}(x_i)}+1]$, the splitting line $\ell(s_i)$ has a y-coordinate in the range $[2n-2i, 2n-2i+1]$.
(ii) The splitting line $\ell(s_i)$ intersects all rectangles R_k with $0 \leqslant k \leqslant \text{rank}(x_i)$.
(iii) The splitting line $\ell(s_i)$ does not intersect any rectangle R_k with $k > \text{rank}(x_i)$.

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 2(ii), the splitting lines $\ell(x_i), \ell(x_j), \ell(x_k)$ all intersect the rectangle $R_{\text{rank}(C)}$. Moreover, by Lemma 2(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. 6, 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.

One important property of our construction is that clause gadgets are only intersected by splitting lines of the variables in the clause. Another important property has to do with the blocking of splitting lines by other splitting lines. Recall that the rank of a variable is the maximum rank of any clause it participates in. We say that a splitting line $\ell(s_i)$ is *blocked* by a splitting line $\ell(s_j)$ if $\ell(s_j)$ intersects $\ell(s_i)$ between s_i and $R_{\text{rank}(x_i)}$. This is dangerous, since it may prevent us from using $\ell(s_i)$ to resolve the cycle in the gadget of a clause containing x_i. The next lemma, proved in the appendix, states the two key properties of our construction.

Lemma 3. *The variable and clause gadgets are placed such that:*
(i) The gadget for any 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 any 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 $j \geqslant i$; the same holds for $\ell(\overline{s_i})$.

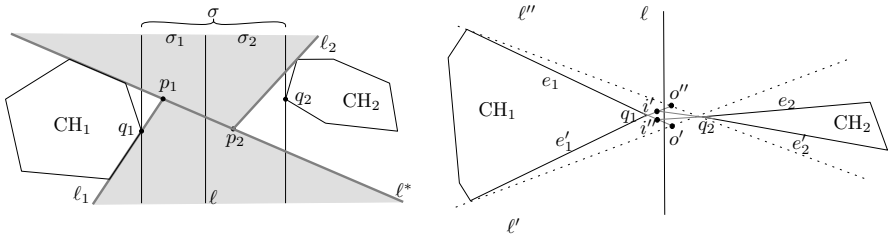

Fig. 7. Left: Illustration for Lemma 4, note that ℓ_1, ℓ_2 and ℓ^* are restricted splitting lines. Right: The case that i, i' are on the left and o' and o'' are on the right side of ℓ

Lemma 3 implies the main result of this section. For the proof see the appendix.

Theorem 2. Perfect Auto-Partition *is* np*-complete.*

3 Optimal free BSPs versus optimal restricted BSPs

Let S be a set of n disjoint line segments in the plane. In this section we will show that $\text{OPT}_{\text{res}}(S) \leqslant 2 \cdot \text{OPT}_{\text{free}}(S)$ for any set S. It follows from the lower bound of Clairbois[4] that this bound is tight.

Consider an optimal free BSP tree $\mathcal{T}$ for S. Let ℓ be the splitting line of the root of $\mathcal{T}$, and assume without loss of generality that ℓ is vertical. Let P_1 be the set of all segment endpoints to the left or on ℓ, and let P_2 be the set of segment endpoints to the right of ℓ. Let CH_1 and CH_2 denote the convex hulls of P_1 and P_2, respectively. We follow the same global approach as Clairbois [4]. Namely, we replace ℓ by a set L of three or four restricted splitting lines that do not intersect the interiors of CH_1 and CH_2, and are such that CH_1 and CH_2 lie in different regions of the partition induced by L. In Fig. 7, for instance, we would replace ℓ by the lines ℓ^*, ℓ_1, ℓ_2. The regions not containing CH_1 and CH_2—the grey regions in Fig. 7—do not contain endpoints, so inside them we can simply make splits along any segments intersecting the regions. After that, we recursively convert the BSPs corresponding to the two subtrees of the root to restricted BSPs. The challenge in this approach is to find a suitable set L, and this is where we will follow a different strategy than Clairbois.

Observe that the segments that used to be cut by ℓ will now be cut by one or more of the lines in L. Another potential cause for extra cuts is that existing splitting lines that used to end on ℓ may now extend further and create new cuts. This can only happen, however, when ℓ crosses the region containing CH_1 and/or the region containing CH_2 in the partition induced by L (the white regions in Fig. 7); if ℓ is separated from these regions by the lines in L, then the existing splitting lines will actually be shortened and thus not create extra cuts. Hence, to prove our result, we will ensure the following properties:

(I) the total number of cuts made by the lines in L is at most twice the number of cuts made by ℓ.

(II) in the partitioning induced by L, the regions containing CH_1 and CH_2 are not crossed by ℓ.

The lines in L are of three types. They are either *inner tangents* of CH_1 and CH_2, or *extensions* of edges of CH_1 or CH_2, or they pass through a vertex of CH_1 (or CH_2) and the intersection of another line in L with a segment in S.

We denote the vertex of CH_1 closest to ℓ by q_1 and we denote the vertex of CH_2 closest to ℓ by q_2 (with ties broken arbitrarily). Let σ be the strip enclosed by the lines through q_1 and q_2 parallel to ℓ, and for $i \in \{1, 2\}$ let σ_i denote the part of σ lying on the same side of ℓ as CH_i—see also Fig. 7. (When q_1 lies on ℓ, then σ_1 will just be a line; this does not invalidate the coming arguments.).

Lemma 4. *Let ℓ^* be a restricted splitting line separating CH_1 from CH_2. Suppose there are points $p_1 \in \ell^* \cap \sigma_1$ and $p_2 \in \ell^* \cap \sigma_2$ such that, for $i \in \{1, 2\}$, the line ℓ_i through p_i and tangent to CH_i that separates CH_i from ℓ is a restricted splitting line. Then we can find a set L of three partition lines satisfying conditions (I) and (II) above.*

Proof. Take ℓ^* as the first splitting line in L. Of all the points p_1 satisfying the conditions in the lemma, take the one closest to $\ell \cap \ell^*$. (If a segment $s \in S$ passes exactly through $\ell \cap \ell^*$, then $p_1 = \ell \cap \ell^*$.) The corresponding line ℓ_1 is the second splitting line in L. The third splitting line, ℓ_2, is generated similarly: of the points p_2 satisfying the conditions of the lemma, take the one closest to $\ell \cap \ell^*$ and use the corresponding line ℓ_2. By construction, p_1p_2 is not intersected by any segment in S, which implies that condition (I) holds. Moreover, ℓ does not cross the regions containing CH_1 and CH_2, so condition (II) holds. □

To show we can always find a set L satisfying conditions (I) and (II), we distinguish six cases. To this end we consider the two inner tangents ℓ' and ℓ'' of CH_1 and CH_2, and look at which of the points q_1 and q_2 lie on which of these lines. Cases (a)–(e) are handled by applying Lemma 4, case (f) needs a different approach. Next we discuss case (a), which is representative for the first five cases. Due to the lack of space, all other cases, including case (f), are discussed in the appendix.

Case (a): Both ℓ' and ℓ'' do not contain any of q_1, q_2. Let e_1 and e_2 be the edges of CH_1 and CH_2 incident to and below q_1 and q_2 respectively. Let $\ell(e_1)$ and $\ell(e_2)$ be the lines through these edges, and assume without loss of generality that $\ell(e_1) \cap \ell(e_2) \in \sigma_1$. We can now apply Lemma 4 with $\ell^* = \ell(e_2)$, and $p_1 = \ell(e_1) \cap \ell(e_2)$, and $p_2 = q_2$. That we can always replace ℓ with a set of segments such that both conditions (I) and (II) holds. As shown in the appendix the same holds for cases (b)-(f) which leads to the following theorem.

Theorem 3. *For any set S of disjoint segments in the plane, $\mathrm{OPT}_{\mathrm{res}}(S) \leqslant 2 \cdot \mathrm{OPT}_{\mathrm{free}}(S)$.*

References

1. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications (3rd edition)*. Springer Verlag, 2008.
2. M. de Berg, M.M. de Groot and M.H. Overmars. Perfect binary space partitions. *Comput. Geom. Theory Appl.* 7:81–91 (1997).
3. M. de Berg, E. Mumford, and B. Speckmann. Optimal BSPs and rectilinear cartograms. In *Proc. 14th Int. Symp. Advances Geographic Inf. Syst. (ACM-GIS)*, pages 19–26, 2006.
4. X. Clairbois. *On Optimal Binary Space Partitions*. MSc thesis, TU Eindhoven, 2006.
5. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.
6. D.E. Knuth and A. Raghunathan. The problem of compatible representatives. *Discr. Comput. Math.* 5:422–427 (1992).
7. D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.* 11:329–343 (1982).
8. C.D. Tóth, Binary space partitions: recent developments. In: J.E. Goodman, J. Pach, and E. Welzl (eds.), *Combinatorial and Computational Geometry*, MSRI Publications Vol. 52, Cambridge University Press, pages 525–552, 2005.
9. B. Katz, I. Rutter, and G. Woeginger. An Algorithmic Study of Switch Graphs. *Lecture Notes in Computer Science*, Springer-Verlag, Volume 5911/2010, pages 226–237, 2010.

Appendix A: Omitted proofs

Lemma 1. $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is satisfiable.

Proof. Suppose there is a truth assignment to the variables $x_1, \ldots, x_m$ that satisfies $\mathcal{C}$. Now consider $\mathcal{C}'$, which is defined over $\{x_1, \ldots, x_m\} \cup \{a, b\}$. Use the same truth assignment for $x_1, \ldots, x_m$, and set $a := \overline{x_i}$ and $b := x_i$. One easily checks that with this truth assignment to a and b, all new and modified clauses are satisfied.

Conversely, suppose there is a truth assignment to $\{x_1, \ldots, x_m\} \cup \{a, b\}$ that satisfies $\mathcal{C}'$. We claim that using the same assignment for $x_1, \ldots, x_m$ will satisfy $\mathcal{C}$. Indeed, $(x_i \vee a) \wedge (\overline{x_i} \vee \overline{a}) \wedge (a \vee b) \wedge (\overline{a} \vee \overline{b})$ implies that $x_i = \overline{a} = b$. This means that C_j (where $\overline{x_i}$ was replaced with a) and all clauses in $\mathcal{C}$ where x_i was replaced with b, are satisfied. All other clauses in $\mathcal{C}$ appear unchanged in $\mathcal{C}'$, and hence, they are also satisfied. □

Lemma 2. Let x_i be a variable, and $\ell(s_i)$ be the splitting line containing s_i.

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

Proof. Observe that

$$(2n - 1 + d_{\mathrm{rank}(x_i)} + 1 - (2i - 2)) \cdot \frac{1}{2d_{\mathrm{rank}(x_i)}} \;\leqslant\; \frac{1}{2} + \frac{2n - 2i + 2}{2d_{\mathrm{rank}(x_i)}} \;<\; 1,$$

since $i \geqslant 1$ and $d_{\mathrm{rank}(x_i)} \geqslant d_0 = 4n$. Hence, the increase in y-coordinate of $\ell(s_i)$ in the x-interval $[2i - 2 : 2n - 1 + d_{\mathrm{rank}(x_i)} + 1]$ is less than 1, proving (i). Property (ii) immediately follows from (i). To prove (iii) we observe that

$$(2n - 1 + d_{\mathrm{rank}(x_i)+1} - (2i - 2)) \cdot \frac{1}{2d_{\mathrm{rank}(x_i)}} \;\geqslant\; \frac{d_{\mathrm{rank}(x_i)+1}}{d_{\mathrm{rank}(x_i)}} \;=\; 2n,$$

so the increase in y-coordinate is at least $2n$ by the time $R_{\mathrm{rank}(x_i)+1}$ is reached. Hence, $\ell(s_i)$ passes above R_k for all $k > \mathrm{rank}(x_i)$. □

Lemma 3. The variable and clause gadgets are placed such that the following holds:

(i) The gadget for any 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 any 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 $j \geqslant i$; the same holds for $\ell(\overline{s_i})$.

Proof. To prove (i), consider a positive clause $C = (x_i \vee x_j \vee x_k)$ with $i < j < k$; the proof for positive clauses with two variables and for negative clauses is similar. The lines $\ell(s_i)$, $\ell(s_j)$, and $\ell(s_k)$ intersect the gadget for C by construction.

Now consider any splitting line $\ell(s_l)$ with $l \notin \{i, j, k\}$. If $\text{rank}(x_l) < \text{rank}(C)$, then $\ell(s_l)$ does not intersect the gadget for C by Lemma 2(iii). If $\text{rank}(x_l) \geqslant \text{rank}(C)$ and $l < i$ or $l > k$, then $\ell(s_l)$ intersects $R_{\text{rank}(C)}$ but not in between $\ell(s_i)$ and $\ell(s_k)$, by Lemma 2(i). Hence, in this case $\ell(s_l)$ does not intersect the clause gadget for C. The remaining case is that $\text{rank}(x_l) \geqslant \text{rank}(C)$ and $i < l < k$. But this is impossible, since the planarity of the embedding implies that if $i < l < k$ and $l \neq j$, then x_l can only participate in clauses enclosed by C, so $\text{rank}(x_l) < \text{rank}(C)$. Finally, we note that the gadget for C obviously is not intersected by any splitting line $\ell(\overline{s_l})$, nor by any splitting line of a segment used in any other clause gadget.

To prove (ii), consider a splitting line $\ell(s_i)$; the proof for a splitting line $\ell(\overline{s_i})$ is similar. If $\ell(s_i)$ is blocked by some $\ell(\overline{s_j})$ then the diagonal placement of the variable gadgets (see Fig. 5) immediately implies $j \geqslant i$. Now suppose that $\ell(s_i)$ is intersected by some $\ell(s_j)$ with $j < i$. Then the slope of $\ell(s_i)$ is greater than the slope of $\ell(s_j)$. This implies that $\text{rank}(x_i) < \text{rank}(x_j)$. Hence, by Lemma 2(i) the intersection must be after $R_{\text{rank}(x_i)}$, proving that $\ell(s_i)$ is not blocked by $\ell(s_j)$. □

Theorem 2. 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 an instance of Planar Monotone 3-SAT with a set $\mathcal{C}$ of m clauses defined over the variables $x_1, \ldots, x_n$. Apply the reduction described above to obtain a set S of $2n + 4m$ segments forming an instance of Perfect Auto-Partition. Note that 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 needed to describe the instance is polynomial in $n + m$. It remains to show that $\mathcal{C}$ is satisfiable if and only if S has a perfect auto-partition.

Suppose S has a perfect auto-partition. Set $x_i :=$ TRUE if s_i is extended before $\overline{s_i}$ in this perfect auto-partition, and set $x_i :=$ FALSE otherwise. Consider a clause $C \in \mathcal{C}$. Since the auto-partition is perfect, the cycle in the gadget for C must be broken. By Lemma 3(i) this can only be done by a splitting line corresponding to one of the variables in the clause, say x_i. But then s_i has been extended before $\overline{s_i}$ and, hence, $x_i =$ TRUE and C is true. We conclude that $\mathcal{C}$ is satisfiable.

Now consider a truth assignment to the variables that satisfies $\mathcal{C}$. A perfect auto-partition for S can be obtained as follows. We first consider s_1 and $\overline{s_1}$. 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, then we proceed with s_3 and $\overline{s_3}$, 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 some, say, 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 3(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 use it to resolve the cycle and get a perfect auto-partition. □

Appendix B: Free BSPs versus restricted BSPs: case analysis

Below are the all the cases (up to symmetries) that can arise when replacing a free splitting line by a set of restricted splitting lines. For completeness we also list the two cases that were already discussed in the main text. Fig. 8 shows the cases (a) to (e).

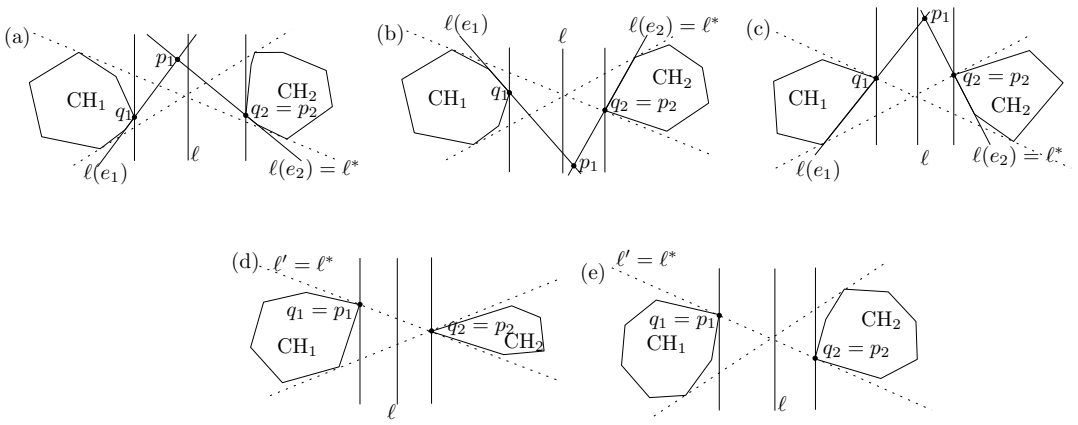

Fig. 8. Illustrations for cases (a)–(e).

Case (a): Both ℓ' and ℓ'' do not contain any of q_1, q_2. Let e_1 and e_2 be the edges of CH_1 and CH_2 incident to and below q_1 and q_2 respectively. Let $\ell(e_1)$ and $\ell(e_2)$ be the lines through these edges, and assume without loss of generality that $\ell(e_1) \cap \ell(e_2) \in \sigma_1$. We can now apply Lemma 4 with $\ell^* = \ell(e_2)$, and $p_1 = \ell(e_1) \cap \ell(e_2)$, and $p_2 = q_2$.

Case (b): ℓ' contains one of q_1, q_2, and ℓ'' does not contain any of q_1, q_2. Assume without loss of generality that the inner tangent ℓ' that has CH_1 below it, contains q_2. We can now proceed as in case (a), except that we let e_1 and e_2 be the edges of CH_1 and CH_2 incident to and *above* q_1 and q_2, respectively.

Case (c): ℓ' contains q_1 and not q_2, and ℓ'' contains q_2 and not q_1. Similar to the previous cases.

Case (d): ℓ' contains both of q_1, q_2, and ℓ'' contains one of q_1, q_2. Apply Lemma 4 with $\ell^* = \ell'$, and $p_1 = q_1$, and $p_2 = q_2$.

Case (e): ℓ' contains both of q_1, q_2, and ℓ'' does not contain any of q_1, q_2. Apply Lemma 4 with $\ell^* = \ell'$, and $p_1 = q_1$, and $p_2 = q_2$.

Case (f): Both ℓ' and ℓ'' contain q_2 but not q_1. This is the most difficult case, and the only one where we need to replace ℓ with four line segments. Let e_1 be the edge incident to and above q_1 and e'_1 the edge incident to and below q_1. Similarly, let e_2 be the edge incident to and above q_2 and e'_2 the edge incident to and below q_2. We denote the intersection of ℓ' and $\ell(e_1)$ by o' and the intersection of ℓ'' and $\ell(e'_1)$ by o''. Also, let $i' = \ell(e'_2) \cap \ell(e'_1)$, and $i'' = \ell(e_2) \cap \ell(e_1)$; see Fig. 7. We consider four subcases below.

Case (f.1): ℓ passes to the right of at least one of o' and o''. Assume without loss of generality that ℓ passes to the right of o'. Now we can apply Lemma 4 with $\ell^* = \ell'$, and $p_1 = o'$, and $p_2 = q_2$.

Case (f.2): ℓ passes to the left of at least one of i' or i''. Assume without loss of generality that ℓ passes to the left of i'. Now we can use Lemma 4 with $\ell^* = \ell(e'_1)$, $p_1 = q_1$ and $p_2 = i'$.

Case (f.3): an input segment intersects $\ell(e_1)$ or $\ell(e'_1)$ in a point $u \in \sigma_2$. Assume without loss of generality $u \in \ell(e_1)$. Now we can apply Lemma 4 with $\ell^* = \ell(e_1)$, $p_2 = u$ and $p_1 = q_1$.

Case (f.4): none of the cases (f.1)-(f.3) applies. As the first splitting line we choose ℓ'. For the second splitting line we initially set $p_1 = o'$ and draw a line $\ell(p_1)$ from p_1 passing through q_1. We move p_1 toward $\ell \cap \ell'$ while moving $\ell(p_1)$ with it, keeping it tangent to CH_1, until it reaches the intersection of an input segment with ℓ'. If we reach such a point before arriving at $\ell \cap \ell'$, we take the resulting line $\ell(p_1)$ as the second splitting line; otherwise we move p_1 back to o' and use that line which is equal to $\ell(e_1)$ as the second splitting line.

For the third splitting line we set $p_2 = \ell' \cap \ell$ and draw a line $\ell(p_2)$ from p_2 that is tangent to CH_2, such that CH_2 lies above it. We move p_2 toward q_2 until it reaches the intersection of a segment with ℓ' (at the end it will reach q_2). For the last splitting line we set $p_3 = \ell(p_1) \cap \ell$ and $p_4 = \ell' \cap \ell$ and make the segment p_3p_4, first we move p_3 toward q_1 until it reaches the intersection of an input segment with $\ell(p_1)$ or q_1. Then, we move p_4 until it reaches the intersection of an input segment with ℓ', if we cannot find such an intersection we rotate p_3p_4 to become fixed by CH_1—see Fig. 9.

It is easy to check that the resulting set, L, of splitting lines satisfies condition (II). It is important to note that p_2 can be on the left or right side of p_1 on ℓ'. When there is at least one input segment intersecting the segment made by o' and $\ell \cap \ell'$, then p_2 is on the left side of p_1 (or the same point as p_1), otherwise p_2 will be on the right side of p_1 and $\ell(p_1) = \ell(e_1)$. To show that condition (I) holds, we first consider the case where p_2 is on the left side of p_1, and then the case where p_2 is on the right side of p_1 is studied.

The segments which are intersected by L have an endpoint in CH_1 and another endpoint in CH_2. Imagine moving along such a segment from its endpoint inside CH_2 to its endpoint inside CH_1. We distinguish two types of segment, depending on whether the first splitting line in L that is crossed is ℓ' or $\ell(p_2)$

A segment of the first type, after intersecting ℓ', can intersect $\ell(p_1)$ or it can intersect p_1p_3 and then p_3p_4. In the first case it intersects two lines of L. To argue about the second case, denote the intersection of p_1p_3 with ℓ by t. By

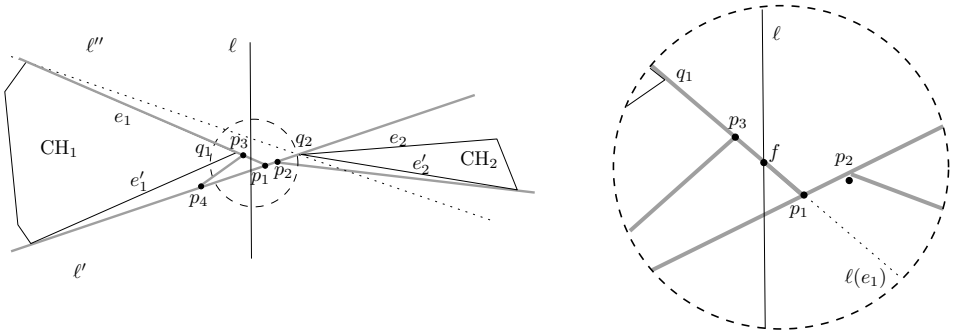

Fig. 9. Illustration of case (f)

construction none of the input segments intersect tp_3, thus the segments of the second type can only intersect p_1t. According to case (f.3) none of the input segments intersects $\ell(e_1)$ in σ_2, thus $o'f$ (which is a part of $\ell(e_2)$ in σ_2) is not intersected by any input segment. By construction p_1o' is not intersected by any input segment. Thus, p_1t is not intersected and there cannot be any segments intersecting ℓ', p_1p_3 and p_3p_4. In conclusion, all input segments of the first type are intersected twice by the lines in L.

Now consider the segments of the second type, which first cross $\ell(p_2)$. After crossing $\ell(p_2)$, they can intersect ℓ', or they can intersect p_2p_4 (a part of ℓ') and then p_3p_4. In the first case, only two lines in L are intersected. As for the second case, by construction we know that none of the input segments intersects p_2p_4. Thus, this case in fact cannot occur. We can conclude that all input segments of the second type are intersected twice by the lines in L.

Now consider the case where p_2 is on the right side of p_1—see Fig. 9. Again we can divide the segments into two types; the segments which first intersect ℓ', and the segments which first intersect $\ell(p_2)$. A segment of the first type can, after intersecting ℓ', intersect $\ell(p_1)$ or it can intersect first p_1p_3 and then p_3p_4. In the first case it intersects two lines of L.

To handle the second case, we denote $\ell(p_1) \cap \ell$ by f. By construction no input segment intersects fp_3, thus the segments of this subset can only intersect p_1f. However, according to case (f.3) none of the input segments intersects $\ell(e_1)$ in σ_2, and we know that in this case $\ell(p_1) = \ell(e_1)$. Hence p_1f, which is a part of it, is not intersected by any input segment, therefore this subset is empty, and all the segments which first intersect ℓ', are intersected twice by the lines in L.

The segments which first intersect $\ell(p_2)$, can then intersect p_2p_1, p_1p_3 and p_3p_4, or they can intersect p_1p_4 and p_3p_4 after intersecting $\ell(p_2)$, or they can intersect only ℓ'. In the last case there are just two lines in L intersected by the segments. By construction we know that none of the input segments intersects p_2p_4, and thus none of them intersects p_2p_1 and p_1p_4. Thus, the first two subsets

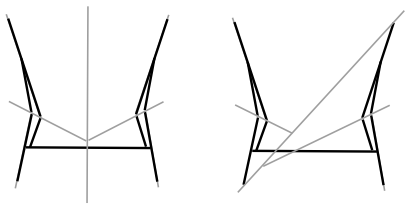

Fig. 10. The structure in which we have $\text{OPT}_{\text{res}}(S) = 2 \cdot \text{OPT}_{\text{free}}(S)$, the black fat segments are input segments and the grey lines are splitting lines. Note that some of the splitting lines contain input segments.

are also empty and the set of input segments in this set are also intersected twice by the lines in L.

A new lower-bound example. Clairbois [4] has shown a construction with a set S of 13 segments for which $\text{OPT}_{\text{res}}(S) = 2$ while $\text{OPT}_{\text{free}}(S) = 1$. That construction shows that our bound is tight. In Fig. 10 a simpler construction is given, which uses only 9 line segments, and for which we also have $\text{OPT}_{\text{res}}(S) = 2$ and $\text{OPT}_{\text{free}}(S) = 1$.