# Faster Algorithm for Minimum Ply Covering of Points with Unit Squares

**Siddhartha Sarkar**

Computer Science and Automation, Indian Institute of Science, Bengaluru



**Abstract.** Biedl et al. introduced the minimum ply cover problem in CG 2021 following the seminal work of Erlebach and van Leeuwen in SODA 2008. They showed that determining the minimum ply cover number for a given set of points by a given set of axis-parallel unit squares is NP-hard, and gave a polynomial time 2-approximation algorithm for instances in which the minimum ply cover number is bounded by a constant. Durocher et al. recently presented a polynomial time $(8+\epsilon)$-approximation algorithm for the general case when the minimum ply cover number is $\omega(1)$, for every fixed $\epsilon > 0$. They divide the problem into subproblems by using a standard grid decomposition technique. They have designed an involved dynamic programming scheme to solve the subproblem where each subproblem is defined by a unit side length square gridcell. Then they merge the solutions of the subproblems to obtain the final ply cover. We use a horizontal slab decomposition technique to divide the problem into subproblems. Our algorithm uses a simple greedy heuristic to obtain a $(27+\epsilon)$-approximation algorithm for the general problem, for a small constant $\epsilon > 0$. Our algorithm runs considerably faster than the algorithm of Durocher et al. We also give a fast 2-approximation algorithm for the special case where the input squares are intersected by a horizontal line. The hardness of this special case is still open. Our algorithm is potentially extendable to minimum ply covering with other geometric objects such as unit disks, identical rectangles etc.

## 1. Introduction

Set Cover is a fundamental problem in combinatorial optimization. Given a range space $(X, \mathcal{R})$ consisting of a set $X$ and a family $\mathcal{R}$ of subsets of $X$ called the ranges, the goal is to compute a minimum cardinality subset of $\mathcal{R}$ that covers all the points of $X$. It is NP-hard to approximate the minimum set cover within a logarithmic factor [Raz and Safra, 1997, Feige, 1998]. When the ranges are derived from geometric objects, it is called the Geometric Set Cover problem. Computing the minimum cardinality set cover remains NP-hard even for simple 2D objects, such as unit squares on the plane [Fowler et al., 1981]. There is a rich literature on designing approximation algorithms for various geometric set cover problems (see [Agarwal and Pan, 2014, Clarkson and Varadarajan, 2007, Hochbaum and Maass, 1985, Chan and Grant, 2014, Mustafa et al., 2014, Erlebach and van

Email: siddharthas1@iisc.ac.in.

Leeuwen, 2010]). More often than not, the geometric versions of the covering problems are efficiently solvable or approximated well. Many variants of the Geometric Set Cover problem find applications in facility location, interference minimization in wireless networks, VLSI design, etc [Demaine et al., 2006, Călinescu et al., 2004]. In this paper, we investigate the following covering problem.

*Problem Statement.* Given a set of geometric objects $\mathcal{S}$, the *ply* of $\mathcal{S}$ is the maximum cardinality of any subset of $\mathcal{S}$ that has non-empty common intersection. A set of objects $\mathcal{S}$ is said to cover a set of points $P$ if for each point $p \in P$ there exists an object $S \in \mathcal{S}$ such that $p$ is contained in $S$. Given a set of objects (for e.g., unit squares) $\mathcal{S}$, and a set of points $P$ on the plane, the goal is to pick a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that every point in $P$ is covered by at least an object in $\mathcal{S}'$ while minimizing the ply.

In this paper we will deal with the minimum ply cover problem where the objects to cover with are unit side length squares.

### 1.1. Our contribution

We design a simple greedy heuristic for axis-parallel unit squares on the plane that achieves an approximation factor of $(27 + \epsilon)$, where $\epsilon > 0$ is a small constant. Our algorithm runs in $O(n^2m^2)$ time where $n$ is the number of input points and $m$ is the number of input squares. Our algorithm is considerably faster when compared to the DP algorithm of [Durocher et al., 2022] that requires $O(n + m^8k^4 \log k + m^8 \log m \log k)$ time, where $k$ is the optimal ply value. Our algorithm is easy to implement and extend to other covering objects.

### 1.2. Related Work

The minimum ply cover problem is a generalization of the minimum membership set cover (i.e., MMSC) problem. Given a set of points $P$ and a family $\mathcal{S}$ of subsets of $P$, the goal in MMSC is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ that covers $P$ while minimizing the maximum number of elements of $\mathcal{S}'$ that contain a common point of $P$. [Kuhn et al., 2005] introduced the general problem and they presented an LP based technique to obtain an approximation factor of $\ln n$ which matches the hardness lower bound. The membership is measured at the points of $P$ in the MMSC problem. [Erlebach and van Leeuwen, 2008] proved that the minimum membership set cover of points with unit disks or unit squares is NP-hard and cannot be approximated by a ratio smaller than 2. For unit squares, they present a 5-approximation algorithm with the assumption that the minimum ply value is bounded by a constant. In the minimum ply cover (abbr. MPC) problem, the ply (i.e., membership) is measured at any point on the plane. [Biedl et al., 2021] prove that the MPC problem is NP-hard for both unit squares and unit disks, and does not admit polynomial-time approximation algorithms with ratio smaller than 2 unless P=NP. They gave polynomial time 2-approximation algorithms for this problem with respect to unit squares and unit disks, when the minimum ply value is bounded by a constant.

## 2. Minimum Ply Covering

Our approach is to divide the problem into distinct subproblems and solve the subproblems. Then we merge the solutions to obtain the ply cover of the original problem. First, let us define and solve a special case of the minimum ply cover problem. The techniques used here will be useful in the future.

### 2.1. Squares are intersected by a horizontal line

Suppose that the input squares are intersected by a horizontal line and the input points lie in at least one of the input squares. Refer to Figure (1). For this special case, we give an algorithm that computes the minimum ply cover of ply at most twice the optimal ply. The algorithm is simple to state and analyze. First, we divide the problem into two subproblems. One corresponding to the input points above the horizontal line and the other corresponding to the input points below the horizontal line.

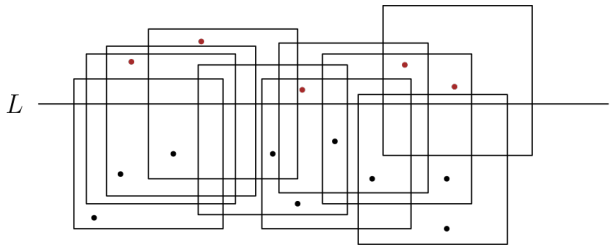


*Figure 1.* An instance of the special case where all the input squares are intersected by a horizontal line $L$ and the points lie on both sides.

**Definition 2.1.** Consider a horizontal line $L$. Now consider a set $\mathcal{S}$ of unit squares intersecting $L$. Let $P$ be a set of points located below $L$ such that each point lies inside at least one of the squares in $\mathcal{S}$. Then **MPCSIHL1** is the problem of computing the minimum ply cover of $P$ with the squares in $\mathcal{S}$.

**MPCSIHL1** stands for **M**inimum **P**ly **C**over for unit **S**quares **I**ntersected by a **H**orizontal **L**ine where points lie on only **1** side (i.e., above or below the horizontal line). Refer to Figure (2)

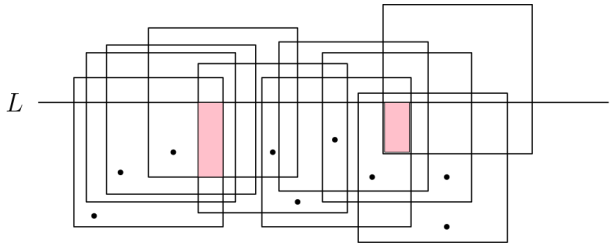


*Figure 2.* An instance of the **MPCSIHL1** problem where all the input points lie below $L$. The pink regions are ply regions with ply value 5.

For a set of squares $\mathcal{S}$, we call a maximum *depth* region on the plane as the *ply region* and denote it by $clique(\mathcal{S})$. So, the *ply region* is the common intersection region below the line $L$. The corresponding depth is the ply which is denoted by $ply(\mathcal{S})$. Note that the *ply region* of a set of squares $\mathcal{S}$ may not be unique. Refer to Figure 2. The Algorithm (1) is a heuristic that computes a minimum ply cover incrementally. The heuristic uses a greedy rule. For multiple sets of squares $S_1, S_2, \ldots, S_k$, we define the following *greedy criteria* for deciding which one to prefer.

*Greedy criteria*. If a set of squares has multiple ply regions, consider the rightmost one as its representative. Use following preference rules to for selection and breaking ties.

1. Prefer the set of squares having the minimum ply value.
2. If rule (1) leads to a tie, prefer the set of squares with the leftmost *ply region* right side.

3. If rule (2) also leads to a tie, prefer the set of squares with the narrowest *ply region*.

4. If the tie persists, then select a candidate set of squares arbitrarily.

**Algorithm 1:** Exact algorithm for MPCSIHL1

**Input :** A horizontal line $L$, a set $\mathcal{S}$ of $m$ unit squares intersecting $L$, a set $P$ of $n$ points below $L$ such that each of them lies in at least one square in $\mathcal{S}$.

**Output :** Returns a set of squares $\mathcal{S}' \subseteq \mathcal{S}$ covering $P$ while minimizing ply.

1 Sort the points in $P$ from left to right.
2 Let the sorted order of points be $p_1, p_2, \ldots, p_n$.
3 $T \leftarrow \infty$ // A 2D array $T$ of dimension $n \times m$ where each entry is initialized to $\infty$ or *infeasible*.
4 **for** $i \leftarrow 1$ **to** $n$ **do**
5  **for** $j \leftarrow 1$ **to** $m$ **do**
6  $T[i,j] = ComputeEntry(i, j, T)$
7  **end**
8 **end**
9 Obtain the best entry in the $n$-th row as per the *greedy criteria*, say $Sol$.
10 **return** $Sol$

The procedure ComputeEntry$(\cdot)$ shows the computation of each entry in the table. Observe that the entry $T[i,j]$ takes into account all the feasible solutions in the $(i-1)$-th row of the table $T$. If $T[i,j]$ is formed by taking the union of $T[i-1,k]$ with $s_j$, for some $k \in [m]$, then we say that $T[i-1,k]$ is the parent of $T[i,j]$ and $T[i,j]$ is derived from $T[i-1,k]$. Intuitively, $T[i,j]$ represents the minimum ply cover for $P_i = \{p_1, \ldots, p_i\}$ as per the *greedy criteria* such that the point $p_i$ is covered by the square $s_j \in T[i,j]$. Starting from $Sol$, it is possible to trace a path to an entry in the first row of the table by following the parents. The path has one node from each row of the table $T$.

After the computation is over, for the $i$-th row in $T$ corresponding to the point $p_i$, there can be multiple entries/covers that achieve the same ply. In the next iteration, while computing $T[i+1,j]$, we will use the *greedy criteria* to select the best entry from the $i$-th row.

Denote by $P_i$ the set $\{p_1, p_2, \ldots, p_i\}$ of the $i$ leftmost input points. The entry $T[i,j]$ is a feasible ply cover for $P_i$, with the constraint that $s_j$ is included in the solution. The algorithm (1) computes a set cover $Sol$ for $P$. We claim that our solution is optimal. We will prove the following **loop invariant** to argue the correctness of our claim.

**Claim 2.2.** *(loop invariant) At the end of the $i$-th iteration of the outer for loop at line 4 of Algorithm (1), in each row $l$ such that $1 \leq l \leq i$, there exists an entry $T(l,j)$ corresponding to the minimum ply set cover for the set of points $P_l$ such that it is the best as per the greedy criteria; $j$ can vary for each row.*

```
1  Function ComputeEntry(i, j, T):
2      if i == 1 then
3          if p_i ∈ s_j then
4              return {s_j}
5          else
6              return ∞
7      Let F_{i-1} be the set of feasible solutions in the (i − 1)-th row, i.e., the entries with finite value.
8      for F ∈ F_{i-1} do
9          Compute the ply region of F ∪ {s_j}
10     Select the best entry among the solutions computed in the for loop in line 9, as per the greedy criteria, say
         Sol_ij.
11     return Sol_ij
```

*Proof.* First, we consider the base case. For the point $p_1$, our algorithm fills the first row of $T$ with feasible solutions containing one square each, i.e., for each square $s_j$ covering $p_1$, the corresponding table entry $T(1, j)$ is a singleton set $\{s_j\}$. The ply of each feasible solution is 1, which is trivially optimal. The entry in row 1 corresponding to the leftmost square containing $p_1$ is the optimal solution for $P_1$ based on our *greedy criteria*.

We fix an $1 < i < n$. Our inductive assumption is that the algorithm has already computed the optimal solution for the first $(i - 1)$ iterations of the outer *for* loop. Specifically, we assume that our *loop invariant* is true after the first $(i - 1)$ iterations.

We need to prove that at the end of the $i$-th iteration of the outer *for* loop, there exists a square $s_j$ containing the point $p_i$ such that the entry $T[i, j]$ is the *optimal* ply cover for $P_i$ satisfying the *greedy criteria*. We term such a solution as the optimal *parametric* solution.

After the $i$-th iteration of the outer *for* loop, let $Sol_i$ denote an optimal set cover (in the sense of the *greedy criteria*) stored at the $i$-th row of the table $T$, for $1 \leq i \leq n$. Suppose for the sake of contradiction, there exists a better solution $OPT_{ij^*}$ for $P_i$ and $p_i \in s_{j^*}$ and $s_{j^*} \in OPT_{ij^*}$. That means either

$$ply(OPT_{ij^*}) < ply(Sol_i) \tag{1}$$

Or

$$ply(OPT_{ij^*}) = ply(Sol_i), \quad \text{but } OPT_{ij^*} \text{ is better than } Sol_i \text{ as per the } \textit{greedy criteria}. \tag{2}$$

Let's consider the possibility (1) first. Our algorithm essentially constructs $Sol_i$ by combining a square $s_j$ with a feasible solution $T(i - 1, j')$ for $P_{i-1}$. If $s_j \in T(i - 1, j')$ or $s_j$ does not intersect the ply region of $T(i - 1, j')$, then $T(i - 1, j') \cup \{s_j\}$ is a feasible solution for $P_i$ with ply $ply(T(i - 1, j'))$.

Let $OPT_{(i-1)j''} \subseteq OPT_{ij^*}$ be a set cover (we do not claim that it has minimum ply) for $P_{i-1}$ with no redundant squares. One way to construct $OPT_{(i-1)j''}$ from $OPT_{ij^*}$ is to simply remove the square (if any) that covers $p_i$

exclusively from $OPT_{ij^*}$. Let $s_{j''} \in OPT_{(i-1)j''}$ and $p_{i-1} \in s_{j''}$. Because of the subset relationship, we have

$$ply(OPT_{(i-1)j''}) \leq ply(OPT_{ij^*}) \tag{3}$$

By our inductive hypothesis, we have

$$ply(Sol_{i-1}) \leq ply(OPT_{(i-1)j''}) \tag{4}$$

Since our algorithm adds at most one square for every new point, we have

$$ply(Sol_i) \leq 1 + ply(Sol_{i-1}) \tag{5}$$

Therefore, from (1), (3), (4), and (5), we conclude that

$$ply(OPT_{(i-1)j''}) = ply(OPT_{ij^*}) = ply(Sol_{i-1}) = ply(Sol_i) - 1$$

Now consider the solutions $OPT_{(i-1)j''}$ for $P_{i-1}$ and $OPT_{ij^*}$ for $P_i$. There are two ways in which $OPT_{ij^*}$ can be related to $OPT_{(i-1)j''}$.

**Case 1**: $\exists$ a square $s_{j^*} \in OPT_{ij^*} \setminus OPT_{(i-1)j''}$ covering $p_i$ such that $s_{j^*}$ does not intersect the ply region of $OPT_{(i-1)j''}$.

**Case 2**: $OPT_{(i-1)j''}$ already covers the point $p_i$ and thus $OPT_{ij^*} = OPT_{(i-1)j''}$.

Figure (3) shows the two cases.

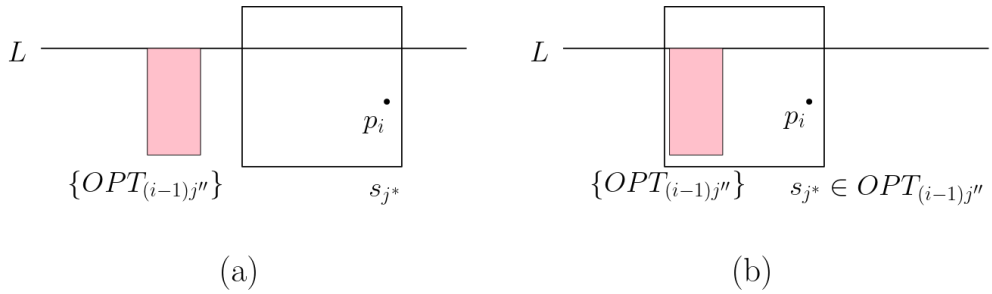


*Figure 3.* The two cases for $OPT_{(i-1)j''}$ with respect to covering the point $p_i$. (a) Case 1, (b) Case 2. The pink region $\{OPT_{(i-1)j''}\}$ refers to the ply region of $OPT_{(i-1)j''}$.

First we consider **Case 1**. Consider a square $s_{j''} \in OPT_{(i-1)j''}$ that covers the point $p_{i-1}$ in $OPT_{(i-1)j''}$. By our inductive hypothesis,

$$ply(Sol_{i-1}) \leq ply(OPT_{(i-1)j''})$$

The square $s_{j^*}$ cannot lie entirely to the left of the left boundary of the rightmost ply region of $OPT_{(i-1)j''}$. Suppose it does. The left boundary of the ply region of $OPT_{(i-1)j''}$ is determined by the rightmost square, say $s_r$, participating in the ply. Since, no square in $OPT_{(i-1)j''}$ is redundant, $s_r$ must cover a point, say $q \in P_{i-1}$, exclusively. Since, by our assumption, $s_{j^*}$ lies to the left of the left boundary of $s_r$, and $p_i$ lies to the right of $q$, hence $s_{j^*}$ cannot cover $p_i$. Hence we have a contradiction. Refer to Figure (4) for an illustration.

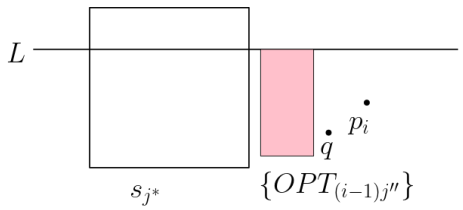


*Figure 4.* The square $s_{j^*} \ni p_i$ lying entirely to the left of the ply region of $OPT_{(i-1)j''}$ is not possible.

Therefore, $s_{j^*}$ must lie entirely to the right of the right boundary of the rightmost ply region of $OPT_{(i-1)j''}$. By virtue of our inductive assumption (about the *greedy criteria* obeyed at every iteration), $Sol_{i-1} \cup \{s_j\}$ is a feasible solution of $P_i$ with ply equal to $ply(Sol_{i-1})$. Therefore,

$$ply(Sol_i) = ply(Sol_{i-1}) \leq ply(OPT_{(i-1)j''}) = ply(OPT_{ij^*})$$

We have arrived at a contradiction. Therefore, **Case 1** is ruled out.

Now we consider **Case 2** where $p_i$ is already covered by $OPT_{(i-1)j''}$. For a square $s$, we write $x_L(s), x_R(s)$ to denote the $x$-coordinates of the left and right boundaries of $s$ respectively. Similarly, write $y_T(s), y_B(s)$ to denote the $y$-coordinates of the top and bottom boundaries of $s$, respectively. We assume that no two squares share the same left boundary. More specifically, the input squares have a left-to-right ordering $\prec$. We write $s_1 \prec s_2$ if $s_1$ lies to the left of $s_2$, i.e., $x_L(s_1) < x_L(s_2)$. If $i < j$, then $p_i$ lies to the left of $p_j$, i.e., $x(p_i) < x(p_j)$.

Let $s_k$ be the rightmost square among the squares covering $p_i$ in $OPT_{(i-1)j''}$. Since $s_k \in OPT_{(i-1)j''}$, there must exist at least one point to the left of $p_i$ which is exclusively covered by $s_k$ in $OPT_{(i-1)j''}$.

Let $p_{i'}$, where $i' < i$, be the leftmost point among the points exclusively covered by $s_k$ in $OPT_{(i-1)j''}$. Since none of the leftmost $(i'-1)$ points is exclusive to $s_k$; hence $OPT_{i'} \setminus OPT_{i'-1} = \{s_k\}$. If $s_k$ contributes to the ply, i.e., $s_k$ intersects the ply region of $OPT_{i'-1}$, then $[OPT_{i'}] = [OPT_{i'-1}] + 1$, else $[OPT_{i'}] = [OPT_{i'-1}]$.

**Claim 2.3.** *The square $s_k$ is the rightmost square in $OPT_{(i-1)j''}$ as per the left to right ordering $\prec$.*

*Proof.* Suppose there exists a square $s_o \in OPT_{(i-1)j''}$ which is rightwards with respect to $s_k$, i.e., $s_k \prec s_o$. The left boundary of $s_o$ must lie to the left of $p_{i-1}$, otherwise $s_o$ will become redundant. If $y_T(s_o) > y_T(s_k)$, then $s_o$ becomes redundant in $OPT_{(i-1)j''}$ as $s_k$ covers the relevant area of $s_o$. The other possibility is $y_T(s_o) < y_T(s_k)$. But the square $s_o$ will contain $p_i$ in this case. This will violate the definition of $s_k$, i.e., $s_k$ being the rightmost among the squares containing $p_i$ in $OPT_{(i-1)j''}$. Hence this is ruled out. Therefore, $s_k$ is the rightmost square in $OPT_{(i-1)j''}$. Refer to Figure (5). □

**Claim 2.4.** *All points between $p_{i'}$ and $p_i$ must lie in the square $s_k$.*

*Proof.* Let there be a point $p_r$ such that $i' < r < i$ and $p_r \notin s_k$. Essentially, $p_r$ lies below the bottom boundary of $s_k$. There must exist a square $s \in OPT_{(i-1)j''}$ covering $p_r$. By our previous claim, $s_k$ is the rightmost square in $OPT_{(i-1)j''}$. Hence $s \prec s_k$. Therefore, $s$ must also cover $p_{i'}$ which is a contradiction since $p_{i'}$ is exclusively covered by $s_k$ in $OPT_{(i-1)j''}$. □

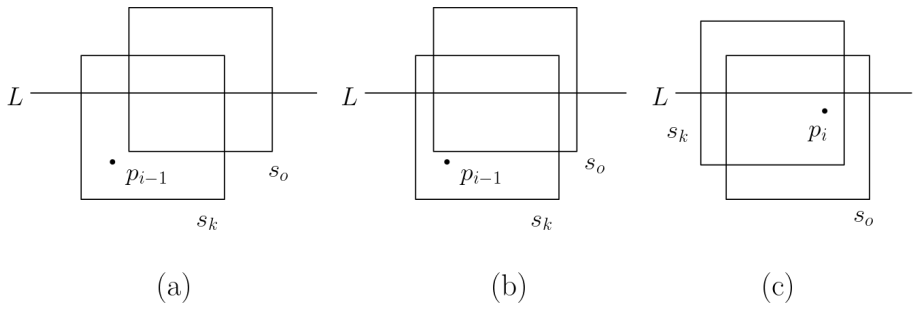


*Figure 5.* Relative positions of $s_k$ and $s_o$. (a) $s_o$ is above $s_k$ but $p_{i-1}$ is to the left of the left boundary of $s_o$. This makes $s_o$ redundant. (b) $s_o$ is above $s_k$ but the entire area of $s_o$ to the left of $p_{i-1}$ is covered by $s_k$. This makes $s_o$ redundant. (c) $s_o$ is below $s_k$, here $s_o$ also covers $p_i$ contradicting the definition of $s_k$. Thus $s_k$ is the rightmost square in $OPT_{(i-1)j''}$.

**Claim 2.5.** *The entry $T(i', k)$ is a feasible set cover for $P_i$ and is as good as the solution $OPT_{ij^*}$. Mathematically, $ply(T(i', k)) = ply(OPT_{ij^*})$.*

*Proof.* By the inductive hypothesis, $ply(Sol_{i'}) \leq ply(OPT_{i'})$. Since $[Sol_{i'}]$ is the optimal ply for covering $P_{i'}$ and due to the definition of our function Compute_Entry$(\cdot)$, we have

$$[Sol_{i'}] \leq [T(i', k)] \leq [Sol_{i'}] + 1 \tag{6}$$

Hence we need to consider only two possible cases: i) $[T(i', k)] = [Sol_{i'}]$ and ii) $[T(i', k)] = [Sol_{i'}] + 1$.

i) $[T(i', k)] = [Sol_{i'}]$: Since all points between $p_{i'}$ and $p_i$ lie in $s_k$ due to Claim (2.4), $T(i', k)$ is a feasible solution for $P_i$. If we picked $s_k$, i.e., $T(i', k)$ for $p_{i'}$ then we are done. If we did not pick $s_k$, then say we picked $s_{k'}$ that does not contain $p_i$. The entry $T(i', k)$ will continue to be present in the table till the $(i-1)$-th row. Eventually, in some $(i'')$-th iteration, the algorithm will select $T(i', k)$ since it covers $p_i$ and does not lead to an increase in ply, where $i' \leq i'' \leq i$. Thus we have a feasible solution $T(i', k)$ in the $(i-1)$-th row of our table till the $i$-th row and $[T(i', k)] \leq [OPT_i]$. Hence $[Sol_i] > [OPT_{ij^*}]$ is ruled out.

ii) $[T(i', k)] = [Sol_{i'}] + 1$: Since $[T(i', k)] \leq [Sol_{i'-1}] + 1$, we have $[Sol_{i'}] = [Sol_{i'-1}]$. In other words, picking $s_k$ for covering $p_{i'}$ will lead to an increase in the current ply value of our solution. Thus $s_k$ participates in the ply of $T(i', k)$. In other words, $s_k$ intersects the ply region of $Sol_{i'-1}$. By our inductive assumption, $\{Sol_{i'-1}\}$ lies relatively to the left of $\{OPT_{i'-1}\}$. Since $s_k$ covers $p_{i'}$ and $p_i$, therefore, $s_k$ also intersects $\{OPT_{i'-1}\}$. Since $OPT_{i'} = OPT_{i'-1} \cup \{s_k\}$, we have $[OPT_{i'}] = [OPT_{i'-1}] + 1$.
So, $[T(i', k)] = [Sol_{i'-1}] + 1 \leq [OPT_{i'-1}] + 1 = [OPT_{i'}]$. If we picked $s_k$, i.e., $T(i', k)$ for $p_{i'}$ then we are done. If we did not pick $s_k$, then suppose we picked $s_{k'}$ that does not contain $p_i$. The entry $T(i', k)$ will continue to be present in the table till the $i$-th row. Eventually, in the $i$-th iteration our algorithm will pick it since it does not lead to an increase in ply. Thus we have $[T(i', k)] \leq [OPT_i]$ in our table. Hence $[Sol_i] > [OPT_{ij^*}]$ is ruled out.

□

This completes the proof of the maintenance of the loop invariant. □

The argument for the possibility (2) is similar. Now consider the running time of Algorithm (1). The sorting of points takes $O(n \log n)$ time. There are $n \cdot m$ entries in $T$ to be computed. For computing the entry $T(i, j)$, all the feasible entries (at most $m$) in row $(i-1)$ need to be considered. Given a set of squares $F$, computing the ply region of $F \cup \{s_j\}$ takes $O(1)$ time. Hence the overall running time is $O(nm^2)$, where $n$ is the number of points and $m$ is the number of squares.

**Theorem 2.6.** *Algorithm (1) is a polynomial time exact algorithm for the **MPCSIHL1** problem (2.1).*

*Proof.* We have already seen that Algorithm (1) runs in polynomial time. Due to the Claim (2.2), the $i$-th row in $T$ contains an optimal solution for $P_i$. Now consider the $n$-th row of the table $T$ corresponding to the point $p_n$. The entry achieving minimum ply is the optimum solution for our input point set $P = P_n$. □

**Theorem 2.7.** *The minimum ply cover for a given set of points lying within the span of a given set of squares intersected by a horizontal line can be approximated within a 2 factor in polynomial time.*

*Proof.* We run the Algorithm (1) twice. Once for the subproblem corresponding to the points lying below the intersecting horizontal line and once for the subproblem corresponding to the points lying above the intersecting horizontal line. Then we return the union of the two solutions, say $Sol_a \cup Sol_b$. Clearly, our solution covers all the input points and hence is a feasible set cover. Now consider an arbitrary point on the plane. It is covered by at most $[Sol_a] + [Sol_b]$ squares, where $ply(Sol_a)$ (resp. $ply(Sol_b)$) is the ply of the problem defined for points above (resp. below) the horizontal line. Since $ply(Sol_a) \leq ply(OPT)$ and $ply(Sol_b) \leq ply(OPT)$, hence our solution is a 2-approximation. □

### 2.2. Points are in a unit height horizontal slab

We consider the special case where the input points lie within a horizontal slab of height 1 and the input squares intersect either the top boundary or the bottom boundary of the slab. Refer to Figure (6). For this case, we give a $(9+\epsilon)$-factor approximation algorithm for computing the minimum ply cover, where $\epsilon > 0$ is a small constant. The algorithm is simple to state and analyze. We define our problem formally.

**Definition 2.8.** Consider two horizontal lines $L_1$ and $L_2$ unit distance apart, where $L_2$ is above $L_1$. Now consider a set $\mathcal{S}$ of unit squares where each square intersects either $L_1$ or $L_2$. Let $P$ be a set of points located below $L_2$ but above $L_1$, each point lying inside at least one of the squares in $\mathcal{S}$. The goal is to compute the minimum ply cover of $P$ with the squares in $\mathcal{S}$.

**Theorem 2.9.** *For unit squares, if there exists a $c$-approximation for the Problem 2.8, then there exists a $3c$-approximation for the minimum ply cover problem.*

*Proof.* We partition the plane into horizontal slabs of unit height. If there are $n$ input points then there are at most $n$ horizontal slabs containing at least 1 point. Suppose we denote the slabs from bottom to top as $H_1, H_2, \ldots, H_n$. We solve for each slab and return the union of the solutions as the final output. Our solution is a feasible solution for all the input points. Consider an arbitrary point on the plane. This point lies within some horizontal slab, say $H_i$. This point may lie in some max clique, i.e., ply region of $H_i$. Simultaneously, this point may lie within the max cliques of the solution for $H_{i-1}$ and $H_{i+1}$. Let $Sol_i$ denote the solution for the slab $H_i$

returned by our algorithm. Let $OPT_i$ denote the optimal solution for the slab $H_i$. Let $Sol = \cup_i Sol_i$. Clearly, $ply(OPT_i) \leq ply(OPT)$ for all $i$, where $OPT$ is the minimum ply for the entire input. Hence

$$\begin{aligned} ply(Sol) &= \max_i (ply(Sol_{i-1}) + ply(Sol_i) + ply(Sol_{i+1}) \\ &\leq c \cdot (ply(OPT_{j-1}) + ply(OPT_j) + ply(OPT_{j+1})) \quad [j : \text{index at which the sum is max.}] \\ &\leq 3c \cdot ply(OPT) \end{aligned}$$

□

We use a similar greedy algorithm as in our previous section. Here the greedy tie-breaking rule is slightly modified as shown below.

- Select the cover with the minimum ply value.
- In case of a tie, prefer a *floating* ply region over an *anchored* ply region.
- In case of a further tie, select the cover with the leftmost ply region right side.
- In case of a further tie, select the cover with the narrowest ply region.
- In case of a further tie, select a cover arbitrarily from the tied covers.

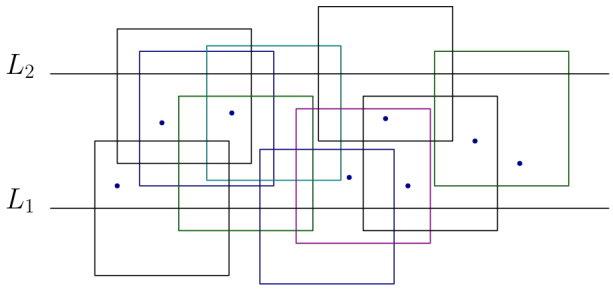


*Figure 6.* The unit height horizontal slab subproblem.

For $1 \leq i \leq n, 1 \leq j \leq m$, we denote the $(i, j)$-th entry of the table as $T(i, j)$. We denote the ply of a minimum ply solution in the $i$-th row by $ply_i$. Recall that by $P_i$, we denote the $i$ leftmost input points. In this section, we will prove the following lemma.

**Lemma 2.10.** *The greedy algorithm is an $(9 + \epsilon)$-approximation algorithm for the problem (2.8).*

**Claim 2.11.** *For any $j$, if the solution $T(i + 1, j)$ is feasible for $P_{i+1}$, then the ply of $T(i + 1, j)$ lies between $ply_i$ and $ply_i + 1$. Notationally,*

$$ply_i \leq ply(T(i + 1, j)) \leq ply_i + 1 \tag{7}$$

*Proof.* The upper bound is straightforward to prove. Since the solution $T(i + 1, j)$ can be composed as the union of the $j$-th square and the minimum ply solution in the $i$-th row; hence the ply of $T(i + 1, j)$ can be at most one

more than $ply_i$.
We prove the lower bound by contradiction. Suppose, for some $j$, the ply of $T(i+1,j)$ is strictly less than $ply_i$. So, $s_j$ does not belong to the optimum solution in the $i$-th row of the table computed by our algorithm. Two cases are possible.
Case 1: While processing the point $p_{i+1}$, $s_j$ can be combined with a minimum ply solution in the $i$-th row, say, $T(i,k)$ such that $T(i,k) \cup \{s_j\}$ is a feasible solution for $P_{i+1}$ and $s_j$ does not intersect the ply region of $T(i,k)$ and at least one square participating in the ply region of $T(i,k)$ is discarded. Thus $ply(T(i+1,j)) < ply_i$.
Case 2: While processing the point $p_{i+1}$, $s_j$ can be combined with a minimum ply solution in the $i$-th row, say, $T(i,k)$ such that $T(i,k) \cup \{s_j\}$ is a feasible solution for $P_{i+1}$ and $s_j$ intersects the ply region of $T(i,k)$ and at least two squares participating in the ply region of $T(i,k)$ are discarded. Thus $ply(T(i+1,j)) < ply_i$.
The ply region of any solution is bound to be a subset of the ply region of its parent solution in the previous row. Hence, in both the cases above, if $s_j$ were a better pick in the $(i+1)$-th row it would have been picked earlier by our greedy algorithm. Hence, we have arrived at a contradiction. □

If a set of squares have a common intersection, we say that they form a clique. We call the common intersection region of the clique as the *ply region*. First, we will classify the cliques in a solution into some distinct types. Let the size of the clique under consideration be $l$, i.e., $l$ squares have a common intersection. If the top side of a square $s_1$ lies above the top side of another square $s_2$, we say that $s_1$ lies above $s_2$. Equivalently, we can say that $s_2$ lies below $s_1$. A set of squares $s_1, s_2, \ldots, s_k$ such that $s_i$ is above $s_{i+1}$ for all $1 \leq i \leq k-1$, is termed as a set of *descending* squares. A set of squares $s_1, s_2, \ldots, s_k$ such that $s_i$ is below $s_{i+1}$ for all $1 \leq i \leq k-1$, is termed as a set of *ascending* squares. If the left side of a square $s_1$ lies to the left of the left side of another square $s_2$, we say that $s_1$ lies to the left of $s_2$. We denote this by $s_1 \prec s_2$. The following types of clique are possible.

- **Top-anchored**: Here all the constituent squares of the clique intersect the top line $L_2$. There are three subtypes as shown in the Figure (7).
  - **Top-anchored ASC**: Here the squares from left to right are ascending. In other words, for any two squares $s_1, s_2$ in the clique such that $s_1 \prec s_2$, the square $s_2$ is above $s_1$.
  - **Top-anchored DESC**: Here the squares from left to right are descending. In other words, for any two squares $s_1, s_2$ in the clique such that $s_1 \prec s_2$, we have $s_2$ below $s_1$.
  - **Top-anchored DESC|ASC**: There is a $k \geq 2$ such that the squares constituting the clique are initially descending from $s_1$ to $s_k$. Then the square $s_{k+1}$ lies above $s_k$. Then the squares $s_{k+1}$ to $s_l$ are ascending. We term the square $s_{k+1}$ as the *transition square*. This type of clique can be viewed as the merger of a descending clique with an ascending clique, in that order.

**Claim 2.12.** *A top anchored clique of type ASC|ASC is forbidden.*

*Proof.* Suppose not. Suppose, the left ascending sequence consists of $k$ squares. Then the rightmost square $s_k$ of the first ASC sequence will become redundant since the second last square $s_{k-1}$ of the first ASC sequence and the transition square $s_{k+1}$ will fully cover the relevant area of $s_k$. This is a contradiction since $s_k$ is redundant. Refer to Figure (8) for an example. □

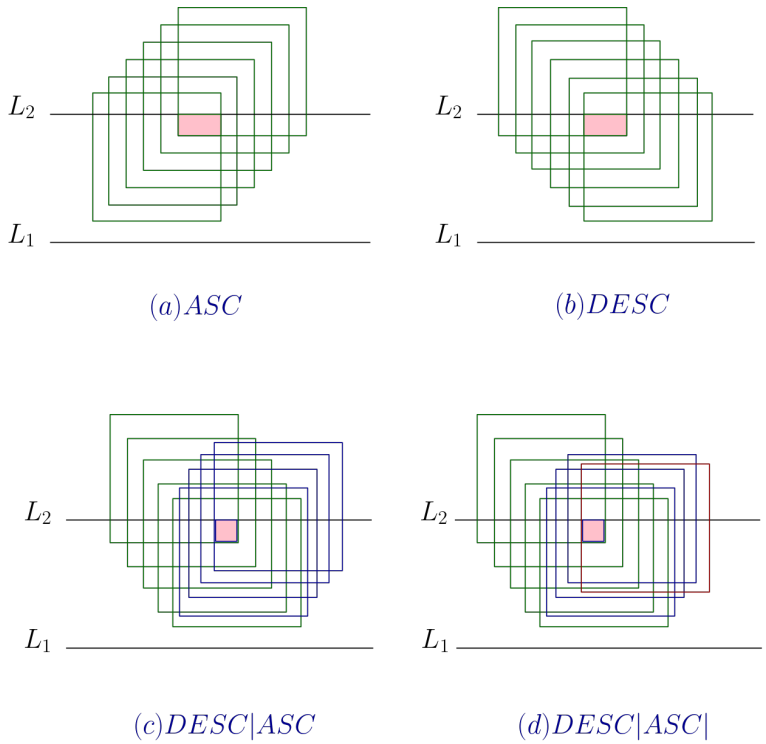


*Figure 7.* (a), (b) and (c) shows three different types of top-anchored cliques. (d) Shows an invalid clique where an DESC|ASC top-anchored clique is followed by a transition square.

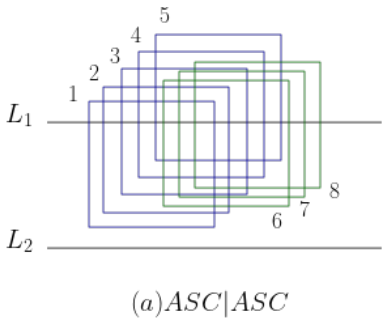


*Figure 8.* A forbidden clique of type top-anchored ASC|ASC. Here the square 5 is redundant since 4 and 6 fully cover the relevant area of 5.

**Claim 2.13.** *A top anchored clique of type DESC|DESC is forbidden.*

*Proof.* Suppose not. Suppose, the left descending sequence consists of $k$ squares. Then the leftmost square $s_{k+1}$ of the second DESC sequence will become redundant since the ast square $s_k$ of the first DESC sequence and the square $s_{k+2}$ will fully cover the relevant area of $s_{k+1}$. This is a contradiction since $s_{k+1}$ is redundant. Refer to Figure 9(a) for an example. □

**Claim 2.14.** *A top anchored clique of type ASC|DESC is forbidden.*

*Proof.* The proof is similar to the proof of Claim (2.12). Refer to Figure 9(b) for an example. □

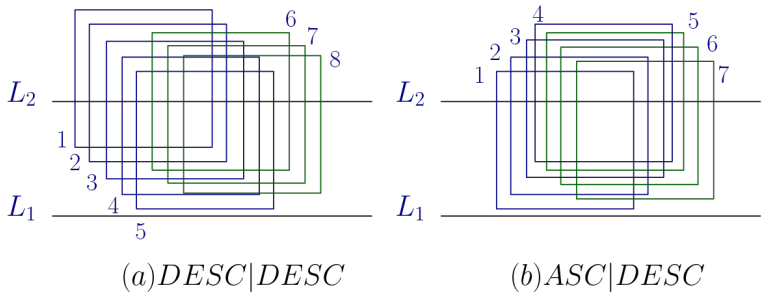


*Figure 9.* (a) A forbidden clique of type top-anchored DESC|DESC. Here the square 6 is redundant since the squares 5 and 7 fully cover the relevant area of 6. (b) A forbidden clique of type top-anchored ASC|DESC. Here the square 4 is redundant since the squares 3 and 5 fully cover the relevant area of 4.

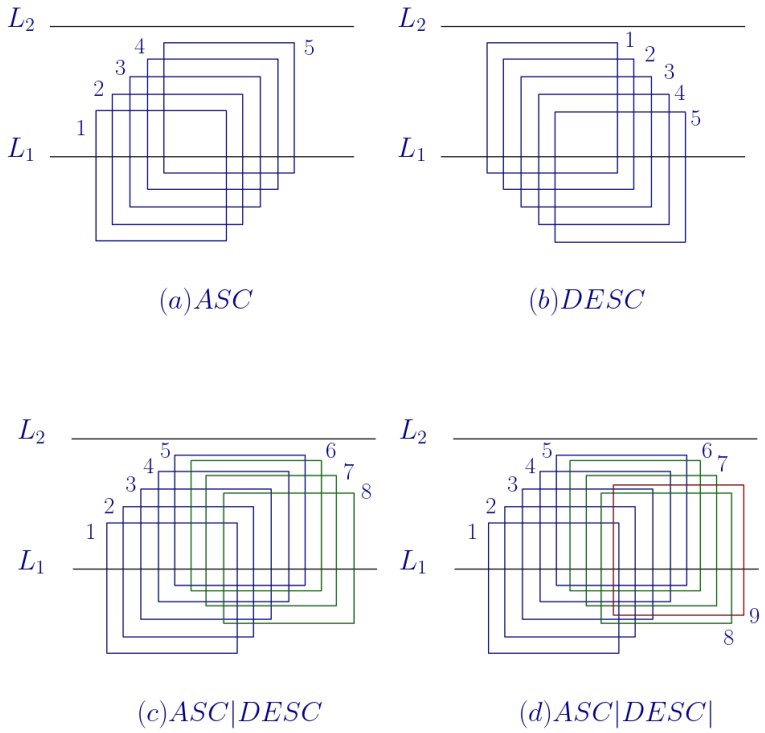


*Figure 10.* (a), (b) and (c) shows three different types of bottom-anchored cliques. (d) Shows an invalid clique where an ASC|DESC bottom-anchored clique is followed by a transition square.

- **Bottom-anchored**: Here all the constituent squares of the clique intersect the bottom line $L_1$. There are three subtypes as shown in the Figure (10).
  - **Bottom-anchored ASC**: Here the squares from left to right are ascending. In other words, for any two squares $s_1, s_2$ in the clique such that $s_1 \prec s_2$, the square $s_2$ is above $s_1$.
  - **Bottom anchored DESC**: Here the squares from left to right are descending. In other words, for any two squares $s_1, s_2$ in the clique such that $s_1 \prec s_2$, the square $s_2$ is below $s_1$.
  - **Bottom anchored ASC|DESC**: There is a $k \geq 2$ such that the squares constituting the clique are initially ascending from $s_1$ to $s_k$. Then the square $s_{k+1}$ lies below $s_k$. Then the squares $s_{k+1}$ to $s_l$ are descending.

**Claim 2.15.** *A bottom anchored clique of type ASC|ASC, DESC|ASC or DESC|DESC is forbidden.*

*Proof.* In all the three cases some squares will become redundant leading to a contradiction. The proof is similar to the proof of Claim (2.12). Refer to Figures (11) and (12) for examples. □

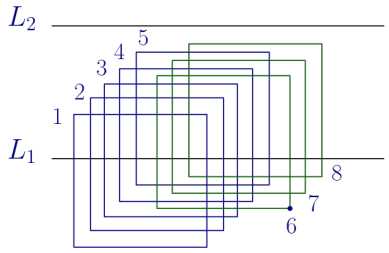


*Figure 11.* A forbidden clique of type bottom-anchored ASC|ASC. Here the square 6 is redundant since the squares 5 and 7 fully cover the relevant area of 6.

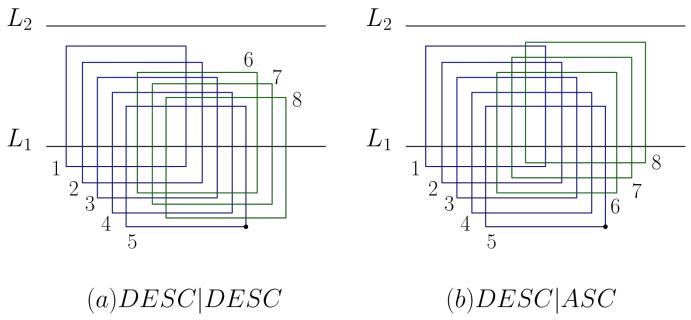


*Figure 12.* (a) A forbidden clique of type bottom-anchored DESC|DESC. Here the square 5 is redundant since the squares 4 and 6 fully cover the relevant area of 5. (b) A forbidden clique of type bottom-anchored DESC|ASC. Here the square 5 is redundant since the squares 4 and 6 fully cover the relevant area of 5.

- **Floating**: If a set of squares have a common intersection and some of the squares intersect the top line $L_2$ while others intersect the bottom line $L_1$, we call the common intersection as a *floating* clique. In the subtypes below, at least one of the squares intersects the bottom line $L_1$ and at least one of the squares intersects the top line $L_2$.
  - **Floating ASC**: Here, the leftmost square $s_1$ intersects the bottom line $L_1$. The squares from left to right are ascending. In other words, for any two squares $s_1, s_2$ in the clique such that $s_1 \prec s_2$, the square $s_2$ is above $s_1$. The rightmost square $s_l$ must intersect the top line $L_2$. Refer to Figure (13(a)).
  - **Floating DESC**: Here, the leftmost square $s_1$ intersects the top line $L_2$. the squares from left to right are descending. In other words, for any two squares $s_1, s_2$ in the clique such that $s_1 \prec s_2$, the square $s_2$ is below $s_1$. The rightmost square $s_l$ must intersect the bottom line $L_1$. Refer to Figure (13(b)).
  - **Floating ASC|ASC**: There is a $k \geq 2$ such that the squares constituting the clique are initially ascending from $s_1$ to $s_k$. Then the square $s_{k+1}$ lies below $s_k$. Then the squares $s_{k+1}$ to $s_l$ are again ascending. Refer to Figure (14). A clique of this type can be thought of as the merger of two monotonic ascending cliques, where the first clique is composed of the squares $s_1$ through $s_k$ and the second clique

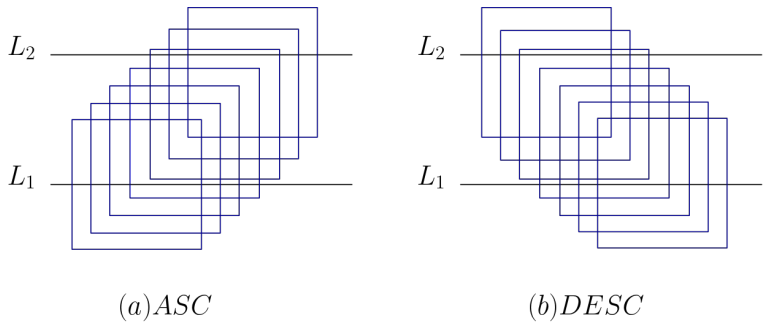


*Figure 13.* Types of monotonic floating cliques.

is composed of the squares $s_{k+1}$ through $s_l$. The structure of such a clique follows certain rules as specified by the following claim.

**Claim 2.16.** *In a floating clique of type ASC|ASC, at most one square of the first ascending sequence can intersect the top line $L_2$ and, at most one square of the second ascending sequence can intersect the bottom line $L_1$.*

*Proof.* Suppose not. There are at least two squares in the first ascending sequence intersecting $L_2$. Then the two rightmost squares in the first ascending sequence $s_{k-1}$ and $s_k$ definitely intersect $L_2$. By definition, $s_{k+1}$ lies below $s_k$. If $s_{k+1}$ intersects the top line $L_2$, then $s_k$ is redundant as $s_{k-1}$ and $s_{k+1}$ cover the relevant area of $s_k$. Refer to Figure 14(a). If $s_{k+1}$ intersects the bottom line $L_1$, then there are two cases. Case 1: $s_{k+2}$ also intersects the bottom line $L_1$, then $s_{k+1}$ is redundant as $s_1, s_k$ and $s_{k+2}$ cover the relevant area of $s_{k+1}$. Refer to Figure 14(b). Case 2: $s_{k+2}$ intersects the top line $L_2$, then $s_k$ is redundant as $s_{k-1}, s_{k+1}$ and $s_{k+2}$ cover the relevant area of $s_k$. Refer to Figure 14(c).
Now consider the second part of the claim. Suppose there are at least two squares in the second ascending sequence intersecting $L_1$. Then its two leftmost squares $s_{k+1}$ and $s_{k+2}$ definitely intersects $L_1$. The square $s_{k+1}$ is redundant as $s_1, s_k$ and $s_{k+2}$ cover the relevant area of $s_{k+1}$. Refer to Figure 14(d). □

– **Floating ASC|DESC**: There is a $k \geq 2$ such that the squares constituting the clique are initially ascending from $s_1$ to $s_k$. Then the square $s_{k+1}$ lies below $s_k$. Then the squares $s_{k+1}$ to $s_l$ are descending. Refer to Figure (15). A clique of this type can be thought of as the merger of a monotonic ascending clique followed by a monotonic descending clique, where the first clique is composed of the squares $s_1$ through $s_k$ and the second clique is composed of the squares $s_{k+1}$ through $s_l$. The structure of such a clique follows certain rules as specified by the following claim.

**Claim 2.17.** *In a clique of type ASC|DESC, at most two squares of the clique can intersect the top line $L_2$.*

*Proof.* Suppose not. There are at least three squares intersecting $L_2$. The square $s_k$ is the topmost square in the clique, hence $s_k$ definitely intersects $L_2$. There are three cases.
Case 1: $s_{k-2}, s_{k-1}, s_k$ intersect $L_2$. The square $s_{k-1}$ is redundant as $s_{k-2}, s_k$ and $s_{k+1}$ cover the relevant area of $s_{k-1}$. Refer to Figure 15(a).

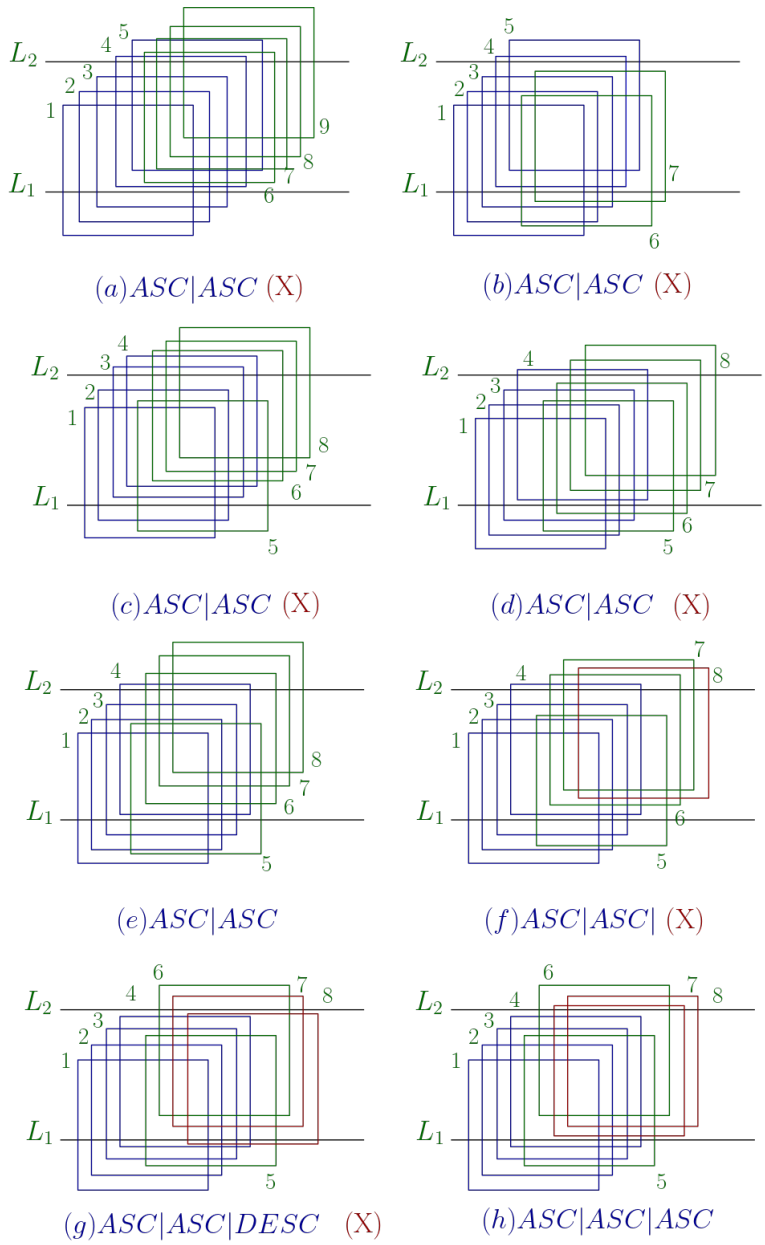


*Figure 14.* Floating ASC|ASC cliques. In (a), (b) and (c), the first ASC sequence has more than 1 square intersecting $L_2$. (a) Shows an invalid clique of ASC|ASC type, where the rightmost square of the first ASC sequence, 5 is covered by 4 and 6. The second ASC sequence has more than 1 squares intersecting $L_1$. (b) Shows an invalid clique of ASC|ASC type where, the transition square 6 is covered by $1, 5$ and 7. (c) Shows an invalid clique of ASC|ASC type where, the rightmost square of the first ASC sequence, 4 is covered by $3, 5$ and 6. (d) Shows an invalid clique where an ASC|ASC the second ascending sequence has more than 1 square intersecting $L_1$. Here, the relevant area of 5 is covered by $1, 4$ and 6. (e) Shows a valid clique of type ASC|ASC. (h) Shows an invalid clique of type ASC|ASC followed by a transition square. (g) Shows an invalid clique of type ASC|ASC|DESC. (h) Shows a valid clique of type ASC|ASC|ASC.

Case 2: $s_{k-1}, s_k, s_{k+1}$ intersect $L_2$. The square $s_k$ is redundant as $s_{k-1}$ and $s_{k+1}$ cover the relevant area of $s_k$. Refer to Figure 15(b).

Case 3: $s_k, s_{k+1}, s_{k+2}$ intersect $L_2$. The square $s_{k+1}$ is redundant as $s_1, s_k$ and $s_{k+2}$ cover the relevant area of $s_{k+1}$. Refer to Figure 15(c).

Thus we have derived a contradiction in each of the cases. Hence proved. □

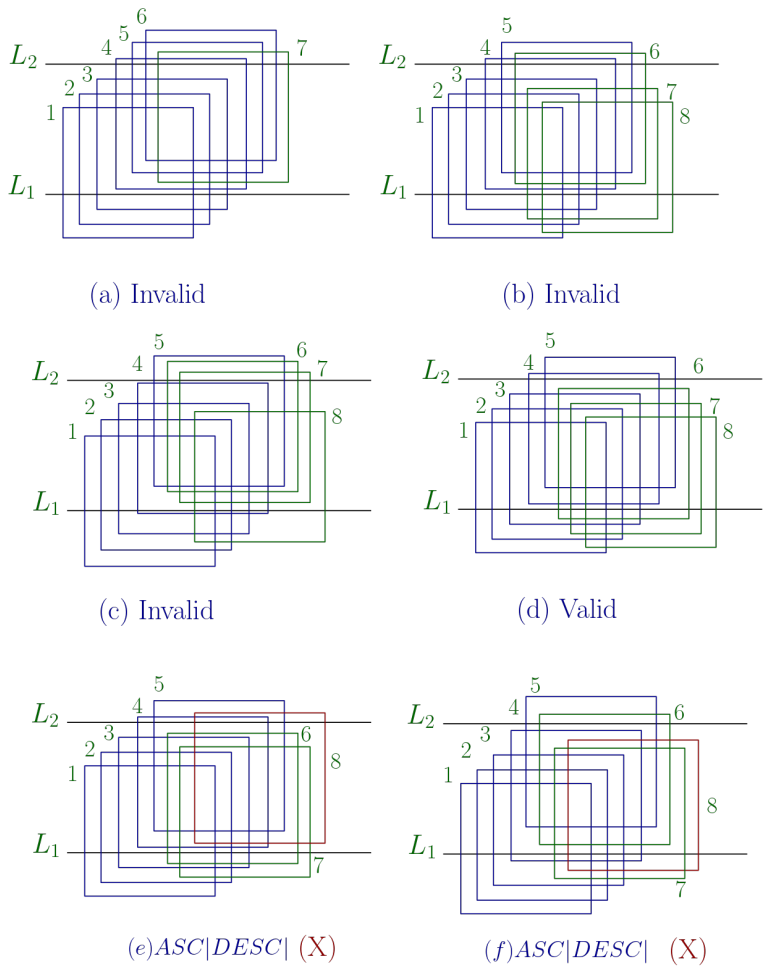


*Figure 15.* (a), (b) and (c) show invalid cliques where more than 2 squares intersect the top line. (a) Here the relevant area of 5 is covered by $4, 6$ and 7. (b) Here the relevant area of 5 is covered by 4 and 6. (c) Here the relevant area of 6 is covered by $1, 5$ and 7. (d) A valid floating ASC|DESC clique where 2 squares intersect the top line. (e) Shows an invalid clique where an ASC|DESC clique is followed by a transition square. The transition square 8 intersects $L_2$. Here the relevant area of 6 is covered by $1, 5, 7$ and 8. (f) Shows an invalid clique where an ASC|DESC clique is followed by a transition square. The transition square 8 intersects $L_1$. Here the relevant area of 7 is covered by $1, 6$ and 8.

– **Floating DESC|ASC**: There is a $k \geq 2$ such that the squares constituting the clique are initially descending from $s_1$ to $s_k$. Then the square $s_{k+1}$ lies above $s_k$. Then the squares $s_{k+1}$ to $s_l$ are ascending. Refer to Figure (16). A clique of this type can be thought of as the merger of a monotonic descending clique with another monotonic ascending clique, where the first clique is composed of the squares $s_1$ through $s_k$ and the second clique is composed of the squares $s_{k+1}$ through $s_l$. The structure of such a clique follows certain rules as specified by the following claim.

**Claim 2.18.** *In a floating clique of type DESC|ASC, at most two squares of the clique can intersect the bottom line $L_1$.*

*Proof.* Suppose not. There are at least three squares intersecting $L_1$. The square $s_k$ is the bottom-most square in the clique, hence $s_k$ definitely intersects $L_1$. There are three cases.
Case 1: $s_{k-2}, s_{k-1}, s_k$ intersect $L_1$. $s_{k+1}$ cannot intersect $L_1$ otherwise $s_k$ will be rendered redundant.

So, $s_{k+1}$ must intersect $L_2$. Now, the square $s_{k-1}$ is redundant as $s_{k-2}, s_k$ and $s_{k+1}$ cover the relevant area of $s_{k-1}$. Refer to Figure 16(a).
Case 2: $s_{k-1}, s_k, s_{k+1}$ intersect $L_1$. The square $s_k$ is redundant as $s_{k-1}$ and $s_{k+1}$ cover the relevant area of $s_k$. Refer to Figure 16(b).
Case 3: $s_k, s_{k+1}, s_{k+2}$ intersect $L_1$. The square $s_{k+1}$ is redundant as $s_1, s_k$ and $s_{k+2}$ cover the relevant area of $s_{k+1}$. Refer to Figure 16(c).
Thus we have derived a contradiction in each of the cases. Hence proved.

□

– **Floating DESC|DESC**: There is a $k \geq 2$ such that the squares constituting the clique are initially descending from $s_1$ to $s_k$. Then the square $s_{k+1}$ lies above $s_k$. Then the squares $s_{k+1}$ to $s_l$ are again descending. Refer to Figure (17). A clique of this type can be thought of as the merger of two monotonic descending cliques, where the first clique is composed of the squares $s_1$ through $s_k$ and the second clique is composed of the squares $s_{k+1}$ through $s_l$. The structure of such a clique follows certain rules as specified by the following claim.

**Claim 2.19.** *In a floating clique of type DESC|DESC, at most one square of the first descending sequence can intersect the bottom line $L_1$ and, at most one square of the second descending sequence can intersect the top line $L_2$.*

*Proof.* Suppose not. There are at least two squares in the first descending sequence intersecting $L_1$. Then the two rightmost squares in the first ascending sequence $s_{k-1}$ and $s_k$ definitely intersect $L_1$. By definition, $s_{k+1}$ lies above $s_k$. Since both $s_{k-1}, s_k$ are intersecting $L_1$, hence $s_1$ must intersect $L_2$, otherwise $s_{k-1}$ will become redundant. If $s_{k+1}$ intersects the bottom line $L_1$, then $s_k$ is redundant as $s_{k-1}$ and $s_{k+1}$ cover the relevant area of $s_k$. Refer to Figure 17(a). On the other hand, if $s_{k+1}$ intersects the top line $L_2$, then there are two cases.
Case 1: $s_{k+2}$ also intersects the top line $L_2$. Then $s_{k+1}$ is redundant as $s_1, s_k$ and $s_{k+2}$ cover the relevant area of $s_{k+1}$. Refer to Figure 17(b).
Case 2: $s_{k+2}$ intersects the bottom line $L_1$, then $s_k$ is redundant as $s_{k-1}, s_{k+1}$ and $s_{k+2}$ cover the relevant area of $s_k$. Refer to Figure 17(c).
Now consider the second part of the claim. Suppose there are at least two squares in the second descending sequence intersecting $L_2$. Then its two leftmost squares $s_{k+1}$ and $s_{k+2}$ definitely intersects $L_2$. The square $s_{k+1}$ is redundant as $s_1, s_k$ and $s_{k+2}$ cover the relevant area of $s_{k+1}$. Refer to Figure 17(d).
Thus we have derived a contradiction in each of the cases. Hence proved. □

**Lemma 2.20.** *If three monotonic sequences of squares $S_1, S_2, S_3$, from left to right respectively, merge to form a clique then $S_2$ consists of at most 2 squares.*

*Proof.* The sequence of squares $S_2$ is either ASC or DESC. We consider the ASC case first. Suppose for the sake of contradiction that there are at least 3 squares in the sequence $S_2$. We denote the three leftmost ones

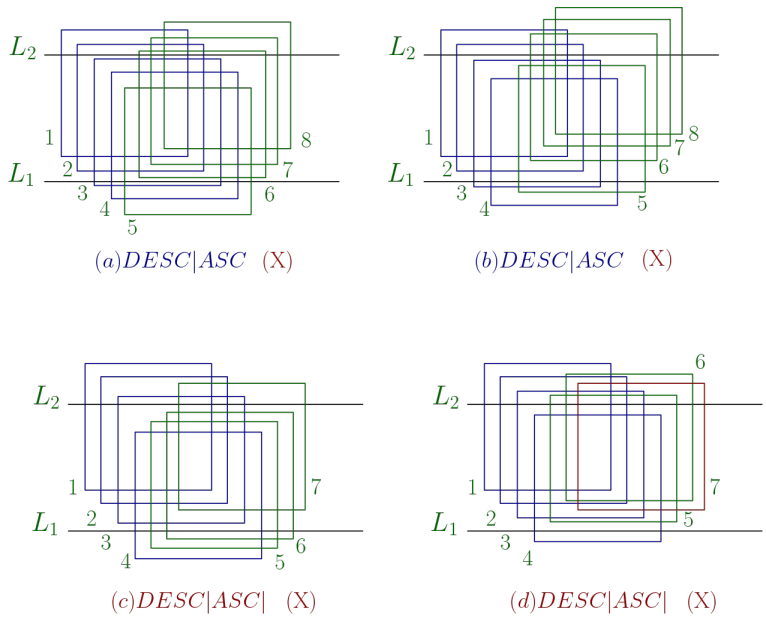


*Figure 16.* (a), (b) and (c) show invalid DESC|ASC cliques where more than 2 squares intersect the bottom line. (a) Here the relevant area of 4 is covered by $3, 5$ and $6$. (b) Here the relevant area of 4 is covered by 3 and 5. (c) Here the relevant area of 5 is covered by $1, 4$ and $6$. (d) Shows an invalid clique where a DESC|ASC clique is followed by a transition square. The transition square 7 will render either square 5 or square 6 redundant.

from left to right as $s_{k+1}, s_{k+2}$ and $s_{k+3}$. We know from the claims (2.15), (2.16) and (2.18) that $S_2$ cannot be a bottom-anchored clique. We have the following possibilities,

i) $S_2$ is top-anchored ASC: Suppose, $|S_1| = k$, $|S_1| + |S_2| = l$. If the transition square of $(S_2, S_3)$, i.e., $s_{l+1}$ is top-intersecting then, $s_l$ will be covered by $s_{l-1}$ and $s_{l+1}$. Thus $s_l$ will become redundant. A contradiction. If $s_{l+1}$ is bottom-intersecting then, the area of $s_{l-1}$ will be covered $s_{k+1}, s_l$ and $s_{l+1}$. Thus $s_{l-1}$ will become redundant. A contradiction.

ii) $S_2$ is floating ASC: If $S_1$ is bottom-anchored DESC then the rightmost square of $S_1$, i.e., $s_k$ will become redundant as its relevant area will be covered by $s_{k-1}$ and $s_{k+1}$. Hence, $S_1$ is either Floating or Bottom-anchored ASC. Now the following cases are possible.

(a) If $s_{k+2}$, $s_{k+3}$ and the transition square of $(S_2, S_3)$, i.e., $s_{l+1}$ are top-intersecting then the relevant area of $s_{k+3}$ will be covered by $s_{k+2}$ and $s_{l+1}$. Thus $s_{k+3}$ will become redundant.
(b) If $s_{k+2}, s_{k+3}$ are top-intersecting but the transition square of $(S_2, S_3)$, i.e., $s_{l+1}$ is bottom-intersecting then there are two cases. Case 1: the transition square of $(S_1, S_2)$, i.e., $s_{k+1}$ is bottom-intersecting, then the relevant area of $s_{k+1}$ will be entirely covered by $s_1$, $s_k, s_{k+2}$ and $s_{l+1}$. Thus $s_1$ will become redundant. Case 2: the transition square between $S_1, S_2$, i.e., $s_{k+1}$ is top-intersecting, then the area of $s_{k+2}$ will be entirely covered by $s_{k+1}$, $s_{k+3}$ and $s_{l+1}$. Thus $s_{k+2}$ will become redundant.
(c) If $s_{k+1}$ and $s_{k+2}$ are bottom-intersecting, $s_{k+3}$ is top-intersecting and the transition square of $(S_2, S_3)$, i.e., $s_{l+1}$) is top-intersecting then there are two cases.
Case 1: The rightmost square of $S_1$, i.e., $s_k$ is bottom-intersecting, then the relevant area of $s_{k+1}$ will be entirely covered by $s_k$ and $s_{k+2}$. Thus $s_{k+1}$ will become redundant.

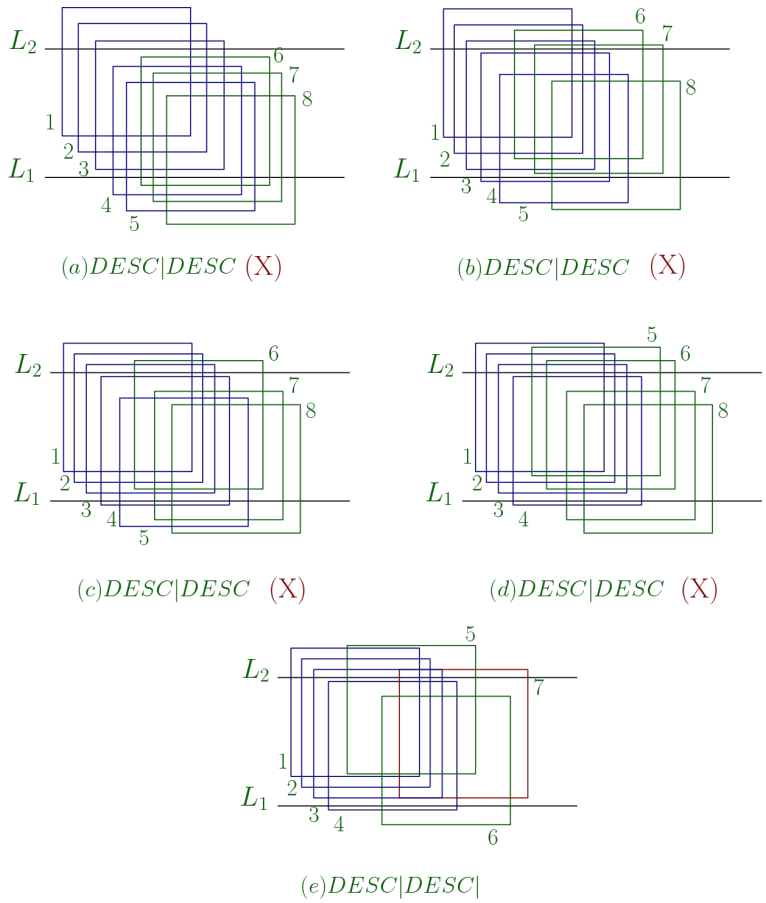


*Figure 17.* (a), (b) and (c) show invalid DESC|DESC cliques where more than 1 square from the first DESC sequence intersect the bottom line $L_1$. (a) Here the relevant area of 5 is covered by 4 and 6. (b) Here the relevant area of 6 is covered by $1, 5$ and 7. (c) Here the relevant area of 5 is covered by $4, 6$ and 7. (d) Shows an invalid DESC|DESC clique where more than 1 square from the second DESC sequence intersect the top line $L_2$. Here the relevant area of 5 is covered by $1, 4$ and 6. (d) Shows an invalid clique where a DESC|DESC clique is followed by a transition square. The transition square 7 will render either square 5 or square 6 redundant.

Case 2: The rightmost square of $S_1$, i.e., $s_k$ is bottom-intersecting, then the relevant area of $s_{k+3}$ will be entirely covered by $s_k, s_{k+2}$ and $s_{l+1}$. Thus $s_{k+3}$ will become redundant.

(d) If $s_{k+1}$ and $s_{k+2}$ are bottom-intersecting, $s_{k+3}$ is top-intersecting and the transition square of $(S_2, S_3)$, i.e., $l_{l+1}$ is bottom-intersecting then there are two cases.

Case 1: the transition square between $S_1, S_2$, i.e., $s_{k+1}$ is bottom-intersecting, then the area of $s_{k+2}$ will be entirely covered by $s_{k+1}$ and $s_{k+3}$. Thus $s_{k+2}$ will become redundant.

Case 2: the transition square between $S_1, S_2$, i.e., $s_{k+1}$ is top-intersecting, then the area of $s_{k+3}$ will be entirely covered by $s_{k+1}, s_{k+2}, s_{k+4}$, and $s_{k+5}$. Thus $s_{k+3}$ will become redundant.

We have shown a contradiction for each of the possibilities when $S_2$ is a sequence of ascending type. Similar arguments are applicable when $S_2$ is a sequence of descending squares. □

**Claim 2.21.** *Our algorithm irrevocably chooses a square $s$ at a point at or to the left of the rightmost exclusive point of $s$.*

*Proof.* Consider a square $s \in Sol$. The square $s$ must have been included into $Sol$ during processing some point $p \in s$. Let $p_r$ be the rightmost exclusive point of $s$ in $Sol$. Suppose $s$ does not exist in the partial solution obtained for the points till the point $p_r$. Then $s$ must have been included at some point $p_q$ to the right of $p_r$. While processing the point $p_q$, the algorithm must have discarded some square(s) so that $p_r$ and other points in $Excl(s)$ can become exclusive to $s$. The resulting solution is a feasible solution for $P_q$. This means that if $s$ was a better pick for $p_q$, it would have been picked earlier by our greedy algorithm. Hence, we have arrived at a contradiction. □

**Claim 2.22.** *When a clique is considered separately, the exclusive regions of all non-extreme squares in the clique are rectangular or L-shaped. All except at most two non-extreme squares may have two different connected exclusive regions.*

*Proof.* Consider any non-extreme square $s_i$. The square $s$ has a square $s_{i-1}$ to its left and a square $s_{i+1}$ to its right. There are $4$ cases.
i) All three squares are in ASC order. Then the exclusive regions of $s_i$ must lie around its top left corner and/or its bottom right corner.
ii) All three squares are in DESC order. Then the exclusive regions of $s_i$ must lie around its top right corner and/or its bottom left corner.
iii) $s_i$ is below both $s_{i-1}$ and $s_{i+1}$: Then $s_i$ has only one exclusive region which is either rectangular or $L$-shaped.
iv) $s_i$ is above both $s_{i-1}$ and $s_{i+1}$: Then $s_i$ has only one exclusive region which is either rectangular or $L$-shaped.
Since the horizontal slab has height 1, hence there can be at most two squares having two different connected exclusive regions. Specifically, in a monotonic DESC clique, the rightmost square intersecting $L_2$ and the leftmost square intersecting $L_1$. And in a monotonic ASC clique, the rightmost square intersecting $L_1$ and the leftmost square intersecting $L_2$. □

Since all the squares in our solution are necessary, hence for every square $s \in Sol$, there exists a set of points $Excl(s)$ such that the points in $Excl(s)$ are contained exclusively in $s$ and no other square in $Sol$. These points in $Excl(s)$ are called exclusive points to $s$. We make the following crucial claim about exclusive points.

**Claim 2.23.** *Let $s_1, s_2$ be two consecutive squares in a maximum clique of Sol such that $s_1 \prec s_2$ and $s_1$ is not the leftmost square in Sol. No input square s can contain all the points in $Excl(s_1) \cup Excl(s_2)$.*

*Proof.* We have already established that any clique in our solution $Sol$ containing no redundant squares can be of only a few types. There are 6 possibilities for the consecutive squares $s_1$ and $s_2$. We analyze them below. In each of the cases below, assume for the sake of contradiction, that there exists a square $s$ such that $s$ covers all the points in $Excl(s_1) \cup Excl(s_2)$.

1. Top Anchored DESC: Here $s_1$ and $s_2$ are intersecting the top line $L_2$ and $s_1 \prec s_2$ and $s_1$ lies above $s_2$. Again there are two subcases.
   (a) If $s$ intersects the bottom line $L_1$: Then at the leftmost exclusive point $p$ of $s_2$, our algorithm has to make a choice between $s$ and $s_2$. Since $s_1$ is already picked due to Claim 2.21, our algorithm will

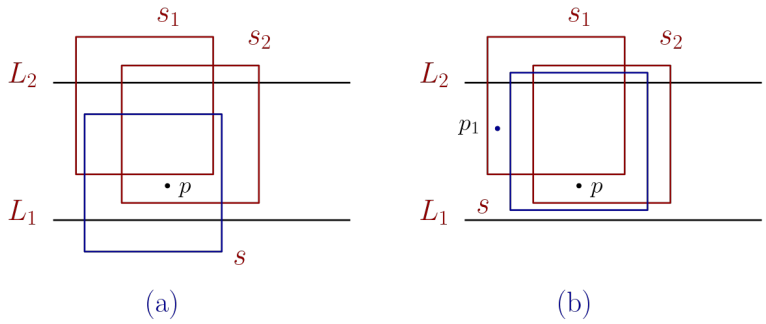


*Figure 18.* The squares $s_1$ and $s_2$ are top anchored and are in descending order in the clique under consideration.

prefer $s$ to $s_2$ since choosing $s_2$ gives a floating clique. And our algorithm would never pick $s_2$ in the future again. Recall that our greedy algorithm prefers floating cliques to anchored cliques. Refer to Figure 18(a) for an illustration.

(b) If $s$ intersects the top line $L_2$: Then at the leftmost exclusive point $p$ of $s_2$, our algorithm would have picked $s$ instead of $s_2$ since picking $s$ would render $s_1$ redundant. Since the exclusive region of $s_1$ is rectangular, and the square $s$ covers all the points in $Excl(s_1)$, hence $s$ lies to the left of $s_2$, i.e., $s \prec s_2$. Therefore, $s$ covers every point in $s_1 \cap s_2$ lying to the left of $p$. Consider a point $p_1 \in s_1 \setminus s_2$, which is covered by another square $s_0 \in Sol$ but presumably not covered by $s$. Clearly, all the exclusive points of $s_0$ lie to the left of $p$. By Claim 2.21, $s_0$ must have been picked by our solution already. Consider a point $p_2 \in s_2 \setminus s_1$, which is covered by another square $s_0 \in Sol$ but presumably not covered by $s$. If $s_2 \prec s_0$ then all the points of $s_0$ lie to the right of $p$. Hence, we need not worry about covering $p_2$ at this stage. Else if $s_0 \prec s_1$ then $s_0 \cap s_2$ will be contained in $s_1 \cap s_2$ and such a point $p_2$ cannot exist. Thus picking $s$ during processing $p$ does not cause an increase in the active ply and our algorithm will pick $s$ greedily. Refer to Figure 18(b) for an illustration.

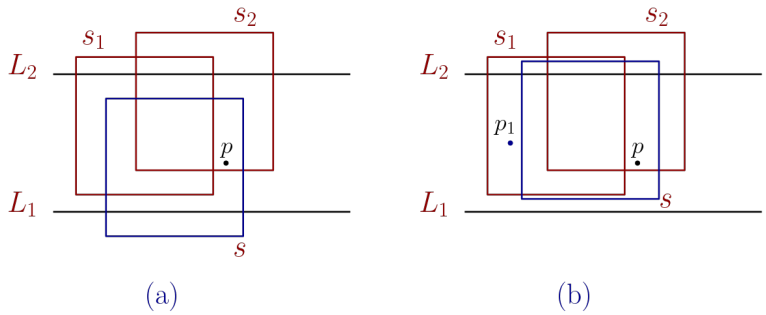


*Figure 19.* The squares $s_1$ and $s_2$ are top anchored and are in ascending order in the clique under consideration.

2. Top Anchored ASC: Here $s_1$ and $s_2$ are intersecting the top line $L_2$ and $s_1 \prec s_2$ and $s_1$ lies above $s_2$. Again there are two subcases.

   (a) If $s$ intersects the bottom line $L_1$: Then at the leftmost exclusive point $p$ of $s_2$, our algorithm would have picked $s$ instead of $s_2$. The reason is exactly same as the argument for the case 1(a) above.

   (b) If $s$ intersects the top line $L_2$: While processing the leftmost exclusive point $p$ of $s_2$, our algorithm has

already picked $s_1$ since the rightmost exclusive point of $s_1$ must lie to the left of $p$. At $p$ our algorithm would have picked $s$ instead of $s_2$ since picking $s$ would also render $s_1$ redundant. There are two possibilities. First, if $s_2 \prec s$, then our algorithm would prefer $s$ to $s_2$ as it would give a narrower clique of same size. Second, if $s_1 \prec s \prec s_2$, then $s$ covers every point in $s_1 \cap s_2$. Consider a point $p_1 \in s_1 \setminus s_2$, which is covered by another square $s_0 \in Sol$ but presumably not covered by $s$. If $s_0 \prec s_1$, then $s_0$ is already picked by our algorithm when we are processing $p$. If $s_2 \prec s_0$, then such a $p$ cannot exist. Refer to Figure 19. Consider a point $p_2 \in s_2 \setminus s_1$, which is covered by another square $s_0 \in Sol$ but presumably not covered by $s$. If $s_2 \prec s_0$ then $s$ covers $p_2$ as $s$ covers $p$. Else if $s_0 \prec s_1$ then such a point $p_2$ cannot exist. Thus picking $s$ during processing $p$ does not cause an increase in the active ply and our algorithm will pick $s$ greedily.

3. Bottom Anchored DESC: $s_1$ and $s_2$ are intersecting the bottom line $L_1$ and $s_1 \prec s_2$ and $s_1$ is above $s_2$. Again there are two subcases.

   (a) If $s$ intersects the top line $L_2$: Then at the leftmost exclusive point $p$ of $s_2$, our algorithm would have picked $s$ instead of $s_2$. The reason is exactly same as the argument for the case 1(a) above.

   (b) If $s$ intersects the bottom line $L_1$: While processing the leftmost exclusive point $p$ of $s_2$, our algorithm has already picked $s_1$ since the rightmost exclusive point of $s_1$ must lie to the left of $p$. At $p$, our algorithm would have picked $s$ instead of $s_2$ since picking $s$ would render $s_1$ redundant. There are two possibilities. First, if $s_2 \prec s$, then our algorithm would prefer $s$ to $s_2$ as it would give a narrower clique of same size. Second, if $s_1 \prec s \prec s_2$, then $s$ covers every point in $s_1 \cap s_2$. Consider a point $p_1 \in s_1 \setminus s_2$, which is covered by another square $s_0 \in Sol$. If $s_0 \prec s_1$, then $s_0$ is already picked by our algorithm when we are processing $p$. If $s_2 \prec s_0$, then such a $p$ cannot exist. Refer to Figure (). Consider a point $p_2 \in s_2 \setminus s_1$, which is covered by another square $s_0 \in Sol$. If $s_2 \prec s_0$ then such a point $p_2$ lies to the right of $p$. Else if $s_0 \prec s_1$ then such a point $p_2$ cannot exist. Thus picking $s$ for $p$ does not cause an increase in the active ply and this conforms to our greedy choice.

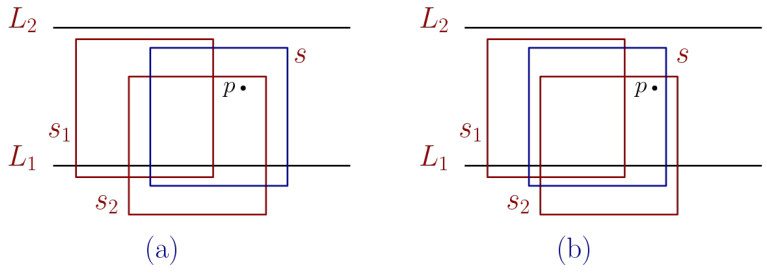


*Figure 20.* The squares $s_1$ and $s_2$ are bottom anchored and are in descending order in the clique under consideration.

4. Bottom Anchored ASC: $s_1$ and $s_2$ are intersecting the bottom line $L_1$ and $s_1 \prec s_2$ and $x_T(s_1) < x_T(s_2)$. Again there are two subcases.

   (a) If $s$ intersects the top line $L_2$: Then at the leftmost exclusive point $p$ of $s_2$, our algorithm has to make a choice between $s$ and $s_2$. Since $s_1$ is already picked, our algorithm will prefer $s$ to $s_2$ since choosing $s_2$ gives a floating clique. And our algorithm would never pick $s_2$ in the future again. Recall that our greedy algorithm prefers floating cliques to anchored cliques.

(b) If $s$ intersects the bottom line $L_1$: Then at the leftmost exclusive point $p$ of $s_2$, our algorithm would have picked $s$ instead of $s_2$ since picking $s$ would render $s_1$ redundant. Since the exclusive region of $s_1$ is rectangular, and the square $s$ covers all the points in $Excl(s_2)$, hence $s$ lies to the left of $s_2$, i.e., $s \prec s_2$. Therefore, $s$ covers every point in $s_1 \cap s_2$ lying to the left of $p$. Consider a point $p_1 \in s_1 \setminus s_2$, which is covered by another square $s_0 \in Sol$. Clearly, all the exclusive points of $s_0$ lie to the left of $p$. By lemma (2.21), $s_0$ must have been picked by our solution already. Consider a point $p_2 \in s_2 \setminus s_1$, which is covered by another square $s_0 \in Sol$. If $s_2 \prec s_0$ then all the points of $s_0$ lie to the right of $p$. Hence, we need not worry about covering $p_2$ at this stage. Else if $s_0 \prec s_1$ then $s_0 \cap s_2$ will be contained in $s_1 \cap s_2$ and such a point $p_2$ cannot exist. Thus picking $s$ during processing $p$ does not cause an increase in the active ply and our algorithm will pick $s$ greedily.

5. Floating DESC: $s_1$ intersects the top line $L_2$ and $s_2$ intersects the bottom line $L_1$ and $s_1 \prec s_2$. Again there are two subcases.

   (a) The square $s$ intersects the top line $L_1$: If $s_2 \prec s$, then at the leftmost exclusive point $p$ of $s_1$, our algorithm would have picked $s$ instead of $s_1$ since picking $s$ also leads to a narrower clique of same size. If $s_1 \prec s \prec s_2$, then at the leftmost exclusive point $p$ of $s_1$, our algorithm would have picked $s$ instead of $s_1$ since picking $s$ leads to a narrower clique of same size. If there exists a point $p_1 \in s_1 \cap s_2$ which is not covered by any other square in $Sol$, then while processing $p_1$, our algorithm will pick $s_2$ instead of $s_1$ since it leads to a narrower clique. Thus our algorithm never picks $s_1$. If $s \prec s_1 \prec s_2$, then at the leftmost exclusive point $p$ of $s_2$, our algorithm would have picked $s$ instead of $s_2$ and discarded $s_1$. If there exists a point $p_1 \in s_1 \cap s_2$ which is not covered by any other square in $Sol$ or by $s$, then while processing $p_1$, our algorithm will pick $s_2$ instead of $s_1$. Thus our algorithm never picks $s_1$.

   (b) The square $s$ intersects the bottom line $L_1$: Exact same arguments as in Case 5(a) are applicable.

6. Floating ASC: $s_1$ intersects the bottom line $L_1$ and $s_2$ intersects the top line $L_2$ and $s_1 \prec s_2$. Similar arguments as presented in the Floating DESC case apply to this case.

Since there are no other possibilities for $s_1, s_2$ and $s$, this completes the proof of our claim. □

**Lemma 2.24.** *Consider one of the maximum cliques, say $K$, in our solution $Sol$. To cover the exclusive points of the squares forming $K$, any feasible set cover has to pick $\lfloor k/3 \rfloor$ squares where $|K| = k$.*

*Proof.* The necessity of $\lfloor k/3 \rfloor$ squares to cover the exclusive points of the squares in the clique $K$ of $Sol$ is a direct consequence of Claim (2.23). We have already shown that the exclusive points in a ASC clique (resp. DESC clique) are monotonically ascending (resp. descending) from left to right except possibly at $4$ squares. First we argue for the case when $K$ is a monotonic clique of ASC type. Consider four consecutive squares in $K$, say, $s_1, s_2, s_3$ and $s_4$, all of which intersect the top line $L_2$. No square $s$ can cover exclusive points from all the $4$ squares. Otherwise such a square $s$ would end up covering all points in $Excl(s_1) \cup Excl(s_2)$. Refer to Figure 21. Similar argument is applicable if the squares are all bottom-intersecting. The only exception can take place if some squares are top-intersecting and some are bottom-intersecting as shown in Figure 22. In this case, a square $s$ may cover exclusive points from all of $s_1, s_2, s_3, s_4$. But covering exclusive points from $5$ squares will

be impossible for similar reasons.

Now partition the squares of the maximum clique into groups of 3 from left to right. For each such group at least 1 square is necessary except possibly for one group at the transition from top-to-bottom or bottom-to-top. Similar arguments apply for transition squares if any. Hence $\lfloor k/3 \rfloor$ squares are necessary. □

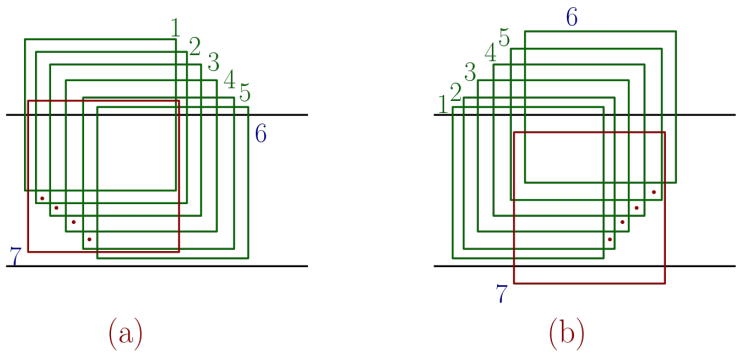


*Figure 21.* (a) Consider the 4 consecutive squares $2, 3, 4, 5$ in this top-anchored monotonic descending clique. The square 7 covers exclusive points from all the four squares. Specifically it covers all the points in $Excl(3) \cup Excl(4)$. (b) Consider the 4 consecutive squares $2, 3, 4, 5$ in this top-anchored monotonic ascending clique. The square 7 covers exclusive points from all the four squares. Specifically it covers all the points in $Excl(3) \cup Excl(4)$.

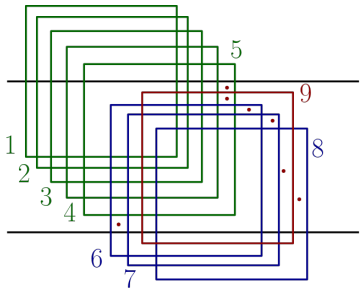


*Figure 22.* The square 9 in this floating clique covers exclusive points from the squares $5, 6, 7, 8$ and covers all exclusive points of only one square, i.e., square 7.

**Lemma 2.25.** *Among the squares required to cover a clique of size $k$ in $Sol$, at least $\lfloor \frac{k}{9} \rfloor - 1$ squares have a common intersection.*

*Proof.* First, consider a top-anchored ASC clique. The exclusive points are also monotonically ascending except possibly at the leftmost square. In this clique, consider a non-extreme exclusive point $p$. Either $p$ can be covered by a top intersecting square from the left or it can be covered by a bottom intersecting square from the right as shown in the Figure 23(b). This implies that all the bottom intersecting squares covering non-extreme exclusive points intersect. Similarly, all the top intersecting squares covering non-extreme exclusive points intersect. Hence, two cliques are formed. Applying the pigeonhole principle, one of the cliques is of size at least $\frac{\lfloor k/3 \rfloor - 2}{2} = \lfloor \frac{k}{6} \rfloor - 1$.

For a monotonic floating clique or a clique which is made up of two monotonic sequences of squares, i.e., a clique of type ASC|DESC or DESC|ASC, the exclusive points may have two different monotonic sequences. Therefore we need a different argument.

Consider a clique $Q$ is of type bottom-anchored ASC|DESC. Observe that there are two monotonic sequences of exclusive points in $Q$ - the first sequence is ascending and the second sequence is descending. As before, two cliques are necessary to cover the points of the first monotonic sequence of exclusive points. Denote these cliques by $Q_1$ and $Q_2$ where $Q_1$ is top-anchored and $Q_2$ is bottom-anchored. Also, there are two other cliques in $OPT$ for covering the exclusive points of the second monotonic sequence. Denote these cliques by $Q_3$ and $Q_4$ where $Q_3$ is bottom-anchored and $Q_4$ is top-anchored. All the $L_1$ (i.e., bottom line) intersecting squares, i.e., the squares in $Q_2$ and $Q_3$ have a common intersection region. Refer to the Figure 23(d). Therefore, in $OPT$ there is exists a clique of size

$$\max(|Q_1|, |Q_2| + |Q_3|, |Q_4|)$$

Applying the pigeonhole principle, one of the cliques have size at least $\frac{\lfloor k/3 \rfloor}{3} - 1 = \lfloor \frac{k}{9} \rfloor - 1$.

Consider a clique $Q$ is of type floating DESC|DESC. Observe that there are two monotonic sequences of exclusive points in $Q$ - both the second sequences are descending. As before, two cliques are necessary to cover the points of the first monotonic sequence of exclusive points. Denote these cliques by $Q_1$ and $Q_2$ where $Q_1$ is bottom-anchored and $Q_2$ is top-anchored. Also, there are two other cliques in $OPT$ for covering the exclusive points of the second monotonic sequence. Denote these cliques by $Q_3$ and $Q_4$ where $Q_3$ is bottom-anchored and $Q_4$ is top-anchored. Here the geometry is such that all the squares in $Q_2$ which are intersecting $L_2$ intersect the squares in $Q_3$, which intersect $L_1$. Refer to the Figure 23(e). Therefore, in $OPT$ there is exists a clique of size

$$\max(|Q_1|, |Q_2| + |Q_3|, |Q_4|)$$

Applying the pigeonhole principle, one of the cliques have size at least $\frac{\lfloor k/3 \rfloor}{3} - 1 = \lfloor \frac{k}{9} \rfloor - 1$. □

**Theorem 2.26.** *Given s set of $n$ points and $m$ axis-parallel unit squares on the plane, our algorithm computes a $(27 + \epsilon)$-factor approximation of the minimum ply cover in $O((nm)^2)$ time, where $\epsilon > 0$ is a small positive constant.*

*Proof.* The approximation factor is a direct consequence of Theorem 2.9 and Lemma 2.25. The algorithm for the horizontal slab subproblem computes a table having $nm$ entries. Each entry can be computed in $O(m)$ time. Therefore, each subproblem requires $O(nm^2)$ time. There are at most $n$ subproblems. Hence the total time required is $O((nm)^2)$. □

## 3. Conclusion

In this paper we have given an algorithmic technique that runs fast for the minimum ply cover problem with axis-parallel unit squares. We have been able to characterize the structure of any clique in our solution and compare it with the maximum clique of the intersection graph of an optimal solution. It may be possible to improve the approximation ratio further. We believe that our technique can be generalized to obtain polynomial-time approximation algorithms for broader class of objects.

**Acknowledgement**. I would like to thank Sathish Govindarajan for many useful discussions on the minimum ply covering problem and for his valuable comments. I would also like to thank Aniket Basu Roy and Shirish Gosavi for their time discussing the problem with me.

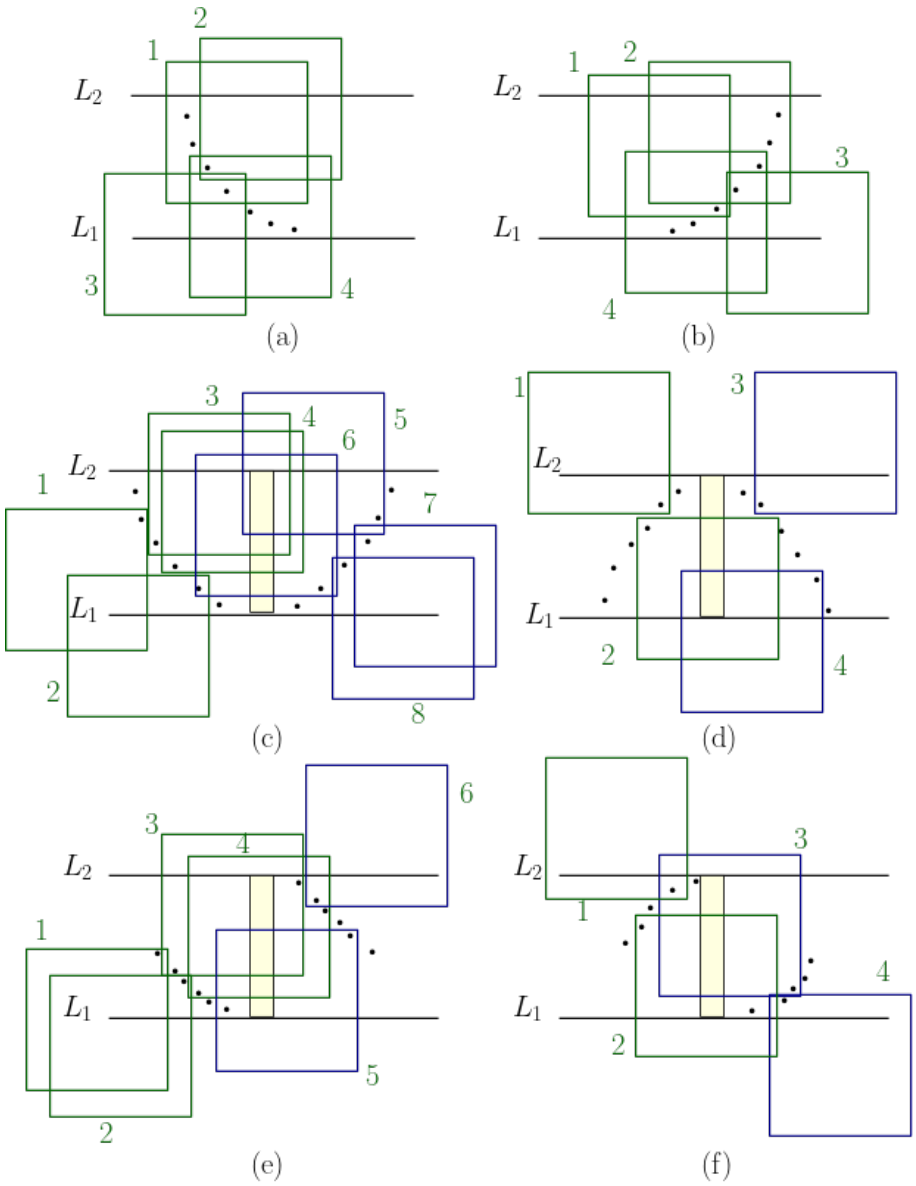


*Figure 23.* Shows the different cases for the proof of Lemma (2.25).

## References

Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, page 271–279, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450325943. doi: 10.1145/2582112.2582152. URL https://doi.org/10.1145/2582112.2582152.

Therese Biedl, Ahmad Biniaz, and Anna Lubiw. Minimum ply covering of points with disks and squares. *Computational Geometry*, 94:101712, 2021. ISSN 0925-7721. doi: https://doi.org/10.1016/j.comgeo.2020.101712. URL https://www.sciencedirect.com/science/article/pii/S0925772120301061.

Gruia Călinescu, Ion I. Măndoiu, Peng-Jun Wan, and Alexander Z. Zelikovsky. Selecting forwarding neighbors

in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):101–111, Apr 2004. ISSN 1572-8153. doi: 10.1023/B:MONE.0000013622.63511.57. URL https://doi.org/10.1023/B:MONE.0000013622.63511.57.

Timothy M. Chan and Elyot Grant. Exact algorithms and apx-hardness results for geometric packing and covering problems. *Computational Geometry*, 47(2, Part A):112–124, 2014. ISSN 0925-7721. doi: https://doi.org/10.1016/j.comgeo.2012.04.001. URL https://www.sciencedirect.com/science/article/pii/S0925772112000740. Special Issue: 23rd Canadian Conference on Computational Geometry (CCCG11).

Kenneth L. Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, Jan 2007. ISSN 1432-0444. doi: 10.1007/s00454-006-1273-8. URL https://doi.org/10.1007/s00454-006-1273-8.

E. D Demaine, Mohammad T. Hajiaghayi, U. Feige, and M. R Salavatipour. Combination can be hard: Approximability of the unique coverage problem. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 162 – 171, 2006/// 2006.

Stephane Durocher, J. Mark Keil, and Debajyoti Mondal. Minimum ply covering of points with unit squares. *CoRR*, abs/2208.06122, 2022. doi: 10.48550/arXiv.2208.06122. URL https://doi.org/10.48550/arXiv.2208.06122.

Thomas Erlebach and Erik Jan van Leeuwen. Approximating geometric coverage problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, page 1267–1276, USA, 2008. Society for Industrial and Applied Mathematics.

Thomas Erlebach and Erik Jan van Leeuwen. Ptas for weighted set cover on unit squares. In Maria Serna, Ronen Shaltiel, Klaus Jansen, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 166–177, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15369-3.

Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, jul 1998. ISSN 0004-5411. doi: 10.1145/285055.285059. URL https://doi.org/10.1145/285055.285059.

Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(81)90111-3. URL https://www.sciencedirect.com/science/article/pii/0020019081901113.

Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, jan 1985. ISSN 0004-5411. doi: 10.1145/2455.214106. URL https://doi.org/10.1145/2455.214106.

Fabian Kuhn, Pascal von Rickenbach, Roger Wattenhofer, Emo Welzl, and Aaron Zollinger. Interference in cellular networks: The minimum membership set cover problem. In Lusheng Wang, editor, *Computing and*

*Combinatorics*, pages 188–198, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31806-4.

Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Settling the apx-hardness status for geometric set cover. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 541–550. IEEE Computer Society, 2014. doi: 10.1109/FOCS.2014.64. URL https://doi.org/10.1109/FOCS.2014.64.

Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 475–484, New York, NY, USA, 1997. Association for Computing Machinery. ISBN 0897918886. doi: 10.1145/258533.258641. URL https://doi.org/10.1145/258533.258641.