Note

A parameterized algorithm for the hyperplane-cover problem✩

Jianxin Wang a, Wenjun Li a, Jianer Chen a,b,∗

a School of Information Science and Engineering, Central South University, Changsha 410083, PR China
b Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843-3112, USA

ARTICLE INFO

Article history:
Received 8 February 2010
Received in revised form 29 June 2010
Accepted 10 August 2010
Communicated by D.-Z. Du

Keywords:
Parameterized algorithm
Computational geometry
Hyperplane-cover
Line-cover

ABSTRACT

We consider the problem of covering a given set of points in the Euclidean space \(\mathbb{R}^m\) by a small number \(k\) of hyperplanes of dimensions bounded by \(d\), where \(d \leq m\). We present a very simple parameterized algorithm for the problem, and give thorough mathematical analysis to prove the correctness and derive the complexity of the algorithm. When the algorithm is applied on the standard hyperplane-cover problem in \(\mathbb{R}^d\), it runs in time \(O^*(k^{(d-1)/d})\), improving the previous best algorithm of running time \(O^*(k^{d+1})\) for the problem. When the algorithm is applied on the line-cover problem in \(\mathbb{R}^2\), it runs in time \(O^*(k^{d/1.35})\), improving the previous best algorithm of running time \(O^*(k^{d/4.84})\) for the problem.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

The line-cover problem is a fundamental problem in geometry, which looks for a minimum number of lines that cover (i.e., contain) all points in a given set in the plane. The problem is NP-hard [6], and also APX-hard to approximate [4]. A parameterized version of the problem, named line-cover, is, for a given set \(S\) of points in the plane and a parameter \(k\), to decide if there are \(k\) lines in the plane that cover all points in \(S\). The line-cover problem can be generalized to the hyperplane-cover problem in a general Euclidean space \(\mathbb{R}^d\), \(d \geq 2\), which is to decide if there are \(k\) (affine) hyperplanes that cover all points in a given set in \(\mathbb{R}^d\) [5].

In this paper, we study a generalized version of the problem in which an additional condition is given for the dimensions of the hyperplanes. For this, we first introduce some definitions.

A set \(B = \{p_0, p_1, \ldots, p_d\}\) of points in the Euclidean space \(\mathbb{R}^m\) determines a hyperplane, denoted by \(H[B]\), which consists of all the points of the form

\[
   c_0 p_0 + c_1 p_1 + \cdots + c_d p_d,
\]

where \(c_0, c_1, \ldots, c_d\) are real numbers satisfying the condition \(c_0 + c_1 + \cdots + c_d = 1\). Pick any point, say \(p_0\), in \(B\). It is easy to verify that the set of points

\[
   H[B] - p_0 = \{p - p_0 \mid p \in H[B]\}
\]

is a subspace of \(\mathbb{R}^m\). We define the dimension of the hyperplane \(H[B]\) to be equal to the dimension of the subspace \(H[B] - p_0\).

Therefore, a \(d\)-dimensional hyperplane \(H\) in \(\mathbb{R}^m\) can be determined by any \(d + 1\) points \(p_0, p_1, \ldots, p_d\) in \(H\) such that the \(d\)

✩ This work is supported by the National Natural Science Foundation of China under Grant Nos. 60773111 and 70921001, and by The Doctoral Discipline Foundation of Higher Education Institution of China under Grant 20090162110056.

∗ Corresponding author at: Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843-3112, USA.
Tel.: +1 409 845 4259; fax: +1 409 847 8578.
E-mail address: chen@cs.tamu.edu (J. Chen).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.08.012
vectors \( p_1 - p_0, \ldots, p_d - p_0 \) are linearly independent. Note that for a hyperplane \( H[B] \) in \( \mathbb{R}^m \) determined by a set \( B \) of \( d+1 \) points, it can be tested in time \( O(d^2 m) \) if a given point \( p \) is in \( H[B] \) by solving a linear equation system of \( d+1 \) unknowns.

Now we are ready to formally define our problem.

**d-Hyperplane-Cover** in \( \mathbb{R}^m \): Given a set \( S \) of \( n \) points in the \( m \)-dimensional Euclidean space \( \mathbb{R}^m \), where \( m \geq d \), decide if there exist \( k \) hyperplanes of dimensions bounded by \( d \) that cover all points in \( S \).

In particular, the standard Hyperplane-Cover problem in \( \mathbb{R}^d \) as defined in [5] is the \((d-1)\)-Hyperplane-Cover problem in \( \mathbb{R}^d \), and the Line-Cover problem [6] is the 1-Hyperplane-Cover problem in the plane \( \mathbb{R}^2 \). There has also been research that studies the \( d \)-Hyperplane-Cover problem in the Euclidean space \( \mathbb{R}^n \) where \( d < m - 1 \) (e.g., [3]). For example, it is interesting to determine if a given set of points in the 3D space \( \mathbb{R}^3 \) is covered by \( k \) lines.

Parameterized algorithms [1] for the standard Hyperplane-Cover problem and the Line-Cover problem have been studied recently. Langerman and Morin [5] presented a parameterized algorithm of running time \( O^*(k^{d/2}) \) for the Hyperplane-Cover problem in \( \mathbb{R}^d \) using a bounded search tree method. In particular, the algorithm in [5] implies a parameterized algorithm of time \( O^*(k^{2k+2}) \) for the Line-Cover problem. Grantson and Levcopoulos [2] proposed an improved parameterized algorithm for the Line-Cover problem with time complexity \( O^*(k/2.2^{2k}) = O^*(k^{2k}/4^{4k}) \), which is currently the best result for the Line-Cover problem.

In this paper, we propose a different algorithmic approach and present a very simple algorithm for the \( d \)-Hyperplane-Cover problem. We provide thorough mathematical analysis to prove the correctness and derive the complexity for the algorithm. Our algorithm for the \( d \)-Hyperplane-Cover problem in \( \mathbb{R}^m \), where \( m \geq d \), runs in time \( O^*(k^{d^2}c_d^2) \), where \( c_d = \sqrt{2\pi (d + 1)}/e > 1.3 \). Our algorithm implies improved algorithms for the standard Hyperplane-Cover problem and for the Line-Cover problem. For the standard Hyperplane-Cover problem in \( \mathbb{R}^d \), our algorithm runs in time \( O^*(k^{(d-1)^2}/c_{d-1}^2) \), where \( c_{d-1} = \sqrt{2\pi d}/e > 1.3 \), improving the previous best algorithm of time \( O^*(k^{d^2}/c_d^2) \) [5]. For the Line-Cover problem, our algorithm runs in time \( O^*(k/1.35^k) \), improving the previous best algorithm of time \( O^*(k^{2k}/4^{4k}) \) [2].

### 2. Solving a recurrence relation

Let \( d \) and \( k \) be two positive integers and let \( n_0, n_1, \ldots, n_d \) be \( d+1 \) non-negative integers satisfying \( \sum_{i=0}^d n_i = k \). In this section, we consider a function \( T(n_0, n_1, \ldots, n_d) \) defined by the following recurrence relation and derive a closed formula for the value \( T(k, 0, \ldots, 0) \). To simplify expressions, let \( T_i(n_0, n_1, \ldots, n_d) = T(n_0, n_1, \ldots, n_i - 1, n_{i+1} + 1, n_{i+2}, \ldots, n_d) \) for \( 0 \leq i \leq d - 1 \).

\[
T(n_0, n_1, \ldots, n_d) = \begin{cases} 1 & \text{if } n_d = k; \\ T_0(n_0, n_1, \ldots, n_d) + \sum_{i=1}^{d-1} n_i T_i(n_0, n_1, \ldots, n_d) & \text{if } n_0 > 0; \\ \sum_{i=1}^{d-1} n_i T_i(n_0, n_1, \ldots, n_d) & \text{if } n_0 = 0 \text{ and } n_d < k. 
\end{cases}
\]

Note that if \( n_i = 0 \) for any \( i \), where \( 1 \leq i \leq d - 1 \), then the term \( n_i T_i(n_0, n_1, \ldots, n_d) \) disappears in the summation. Also, if \( n_d = k \) then \( n_i = 0 \) for all \( 0 \leq i \leq d - 1 \), and by definition \( T(0, \ldots, 0, k) = 1 \).

The recurrence relation (1) comes from the following combinatorial problem. Suppose that we have \( dk \) distinct objects that are given in a fixed order \( \{a_1, a_2, \ldots, a_{dk}\} \), and \( k \) undistinguished boxes. We are interested in studying the number of different ways of placing the objects in boxes. Consider the first \( n \) objects \( \{a_1, \ldots, a_n\}, n \leq dk \). A placement of \( \{a_1, \ldots, a_n\} \) is a distribution of the \( n \) objects into the boxes so that no box contains more than \( d \) objects. A placement of \( \{a_1, \ldots, a_n\} \) has a configuration \( \{n_0, n_1, \ldots, n_d\} \) if for every \( i, 0 \leq i \leq d \), there are \( n_i \) boxes that contain exactly \( i \) objects. Note that two different placements of \( \{a_1, \ldots, a_n\} \) may have the same configuration. Moreover, since the boxes are undistinguished, two placements \( P_1 \) and \( P_2 \) are regarded the same if there is a one-to-one mapping from the boxes of \( P_1 \) to the boxes of \( P_2 \) such that the corresponding boxes contain exactly the same set of objects.

Consider the recursive algorithm **Placement** given in Fig. 1.

We make some remarks on the algorithm. First note that in case \( n_0 > 0 \), the algorithm on the given placement of \( \{a_1, \ldots, a_n\} \) has only one branch that adds the object \( a_{n+1} \) to an empty box. All other empty boxes are not considered. Moreover, the algorithm never adds the object \( a_{n+1} \) to a box that already contains \( d \) objects. Thus, each branch of the algorithm will work on a valid placement of the objects \( \{a_1, \ldots, a_{n+1}\} \). Also note that as long as \( n < dk \), there must be some boxes that contain fewer than \( d \) objects. Thus, the algorithm can always proceed. The algorithm only stops when \( n = dk \) and when all boxes contain exactly \( d \) objects.

The execution of the algorithm **Placement** is recursive and can be depicted by a branching tree \( T \). Each non-leaf node \( u \) in \( T \) is associated with a placement of a list \( \{a_1, \ldots, a_{n_u}\} \) of objects for some \( m \geq n \). Each child of the node \( u \) in \( T \) corresponds to a placement of the list \( \{a_1, \ldots, a_{n_u+1}\} \). Each leaf in \( T \) corresponds to a placement of \( \{a_1, \ldots, a_{n_u}\} \).

Let \( T(n_0, n_1, \ldots, n_d) \) be the number of leaves of the branching tree \( T \) for the algorithm **Placement**\((n_0, n_1, \ldots, n_d)\) on a placement of \( \{a_1, \ldots, a_n\} \) of configuration \( \{n_0, n_1, \ldots, n_d\} \). It is fairly easy to verify that the function \( T(n_0, n_1, \ldots, n_d) \) satisfies the recurrence relation (1). In fact, for each \( i, 1 \leq i \leq d - 1 \), and for each of the \( n_i \) boxes that contain exactly \( i \) objects, there is a branch in the algorithm that recursively works on a placement of \( \{a_1, \ldots, a_{n_i}, a_{n_i+1}\} \) of configuration...
Algorithm Placement\( (n_0, n_1, \ldots, n_d) \)

Input: a placement of \( \{a_1, \ldots, a_n\} \) that has a configuration \( (n_0, n_1, \ldots, n_d) \).
Output: extensions of the given placement of \( \{a_1, \ldots, a_n\} \) into placements of \( \{a_1, \ldots, a_k\} \).

1. if \( n = dk \) then stop;
2. branch
   2.1 for each \( i, 1 \leq i \leq d - 1 \), and each box containing \( i \) objects, branch by adding the object \( a_{n+i} \) to the box, and recursively working on the resulting placement of \( \{a_1, \ldots, a_{n+i}\} \);
   2.2 in case \( n_0 > 0 \), also branch by adding the object \( a_{n+1} \) to the next empty box, and recursively working on the resulting placement of \( \{a_1, \ldots, a_{n+1}\} \).\end{quote}

\( (n_0, \ldots, n_{i-1}, n_i - 1, n_{i+1} + 1, n_{i+2}, \ldots, n_d) \). Moreover, if \( n_0 > 0 \), then there is also a branch in the algorithm that recursively works on a placement of \( \{a_1, \ldots, a_n, a_{n+1}\} \) of configuration \( (n_0 - 1, n_1 + 1, n_2, \ldots, n_d) \).

We say that a placement \( P_{dk} \) of \( \{a_1, \ldots, a_k\} \) is an extension of a placement \( P_n \) of \( \{a_1, \ldots, a_n\} \) if \( P_n \) can be obtained from \( P_{dk} \) by removing the objects \( \{a_{n+1}, \ldots, a_{dk}\} \) from the boxes.

**Lemma 2.1.** For every placement \( P_{dk} \) of \( \{a_1, \ldots, a_k\} \) that is an extension of the placement \( P_n \) of \( \{a_1, \ldots, a_n\} \), there is a unique leaf in the branching tree \( T \) of the algorithm Placement on input \( P_n \) that constructs the placement \( P_{dk} \).

**Proof.** By induction on \( h = dk - n \), it is easy to prove that for each placement \( P_{dk} \) of \( \{a_1, \ldots, a_k\} \) that is an extension of the placement \( P_n \), there is a leaf in the branching tree \( T \) of the algorithm Placement on input \( P_n \) that constructs \( P_{dk} \). In the following, we prove that no two leaves in the branching tree \( T \) construct the same placement of \( \{a_1, \ldots, a_k\} \).

Let \( u \) be the least common ancestor of two different leaves \( \mathbf{v} \) and \( \mathbf{w} \) in the tree \( T \), and suppose that \( u \) is associated with a placement \( P_u \) of the list \( \{a_1, \ldots, a_m\} \), \( m \geq n \). When the algorithm Placement is recursively called on the placement \( P_u \), the two branches that lead to the leaves \( \mathbf{v} \) and \( \mathbf{w} \), respectively, put the object \( a_{m+1} \) in different boxes in \( P_u \). In particular, one of the branches must add \( a_{m+1} \) to a non-empty box \( \mathbf{B}' \) and the other branch adds \( a_{m+1} \) to a different (empty or non-empty) box \( \mathbf{B}'' \). Suppose that the box \( \mathbf{B}' \) contains an object \( a_i \) before \( a_{m+1} \) is added, then the two placements are different: after adding \( a_{m+1} \), \( a_i \) and \( a_{m+1} \) are in the same box in one placement and in different boxes in the other placement. Since we never remove objects from boxes during the execution of the algorithm, when the two branches reach the leaves \( \mathbf{v} \) and \( \mathbf{w} \), they must end up with different placements of \( \{a_1, \ldots, a_k\} \). \( \square \)

By Lemma 2.1, the number of leaves in the branching tree \( T \) for the algorithm Placement on a placement \( P_n \) of \( \{a_1, \ldots, a_n\} \) is exactly equal to the number of different placements of \( \{a_1, \ldots, a_k\} \) that are extensions of \( P_n \). Now we are ready for our main result for this section.

**Theorem 2.2.** Let \( d \) and \( k \) be two positive integers and let \( n_0, n_1, \ldots, n_d \) be \( d + 1 \) non-negative integers satisfying \( \sum_{i=0}^{d} n_i = k \).
Let \( T(0, \ldots, 0) = (dk)!/((d!)^k k!) \).

**Proof.** Let \( P_0 \) be the placement of the empty sublist of \( \{a_1, \ldots, a_k\} \) (i.e., all \( k \) boxes are empty, \( n_0 = k \)). Consider the branching tree \( T_0 \) for the algorithm Placement on the placement \( P_0 \). By our discussion, \( T(k, 0, \ldots, 0) \) is the number of leaves of the tree \( T_0 \). Since every placement of \( \{a_1, \ldots, a_k\} \) is an extension of \( P_0 \), by Lemma 2.1, \( T(k, 0, \ldots, 0) \) is also equal to the number of different placements of \( \{a_1, \ldots, a_k\} \).

On the other hand, there is an alternative way to compute the number of different placements of \( \{a_1, \ldots, a_k\} \) as follows. Pick \( d \) objects from \( \{a_1, \ldots, a_k\} \) and put them into one box, then pick another \( d \) objects from the remaining \( dk - d \) objects and put them in the second box, then pick another \( d \) objects from the remaining \( dk - 2d \) objects and put them in the third box, and so on. The number of different such ways to put the \( dk \) objects \( \{a_1, \ldots, a_k\} \) in the \( k \) boxes is equal to

\[
\binom{dk}{d} \cdot \binom{(k-1)d}{d} \cdot \cdots \cdot \binom{2d}{d} \cdot \binom{d}{d}.
\]

By our assumption, the boxes are undistinguished. Thus, each placement \( P_{dk} \) of \( \{a_1, \ldots, a_k\} \) corresponds to \( k! \) different ways to put the objects in the above manner: each permutation of the boxes in \( P_{dk} \) corresponds to a different way. In summary, the total number of different placements of \( \{a_1, \ldots, a_k\} \) is equal to

\[
T(k, 0, \ldots, 0) = \frac{\binom{dk}{d} \cdot \binom{(k-1)d}{d} \cdot \cdots \cdot \binom{2d}{d} \cdot \binom{d}{d}}{k!} = \frac{(dk)!}{(d!)^k k!}.
\]

This completes the proof of the theorem. \( \square \)

3. A simple algorithm for hyperplane-cover

Let \( m \) and \( d \) be fixed positive integers, \( m \geq d \). Now we consider the \( d \)-HYPERPLANE-COVER problem in the Euclidean space \( \mathbb{R}^m \). To simplify expressions, we will instead study the \((d - 1)\)-HYPERPLANE-COVER problem in the Euclidean space \( \mathbb{R}^m \).
Algorithm HyperplaneCover($S'; n_0, n_1, \ldots, n_d$)
Input: a set $S'$ of points in $\mathbb{R}^m$, and a box placement of configuration $(n_0, n_1, \ldots, n_d)$. Output: an extension of the given box placement to $k$ hyperplanes of dimensions bounded by $d$ that cover all points in $S'$ if such an extension exists.
1. if $S' = \emptyset$ then return the input box placement;
2. if $(n_d = k$ and $S' \neq \emptyset$) then return("this branch fails");
3. pick an arbitrary point $p$ from $S'$;
4. branch
4.i for each $i$, $1 \leq i \leq d - 1$, and each box $B$ containing $i$ points do
branch by adding the point $p$ to the box $B$, removing all points in $S'$ that are in the hyperplane $H[B]$, and recursively working on the resulting set and the resulting box placement of configuration $(n_0, n_1, \ldots, n_{i-1}, n_i - 1, n_{i+1} + 1, n_{i+2}, \ldots, n_d)$; 
4.0 if $n_d > 0$ then
also branch by adding the point $p$ to an empty box, and recursively working on the set $S' \setminus \{p\}$ and the resulting box placement of configuration $(n_0 - 1, n_1 + 1, n_2, \ldots, n_d)$.

Our algorithm for the $(d - 1)-$HYPERPLANE-COVER problem is very similar to the algorithm Placement given in the previous section: we use one box for each $(d - 1)$-hyperplane. Initially, all boxes are empty. Inductively, suppose that for each $i$, $0 \leq i \leq d$, there are $n_i$ boxes that contain exactly $i$ points (this will be called a box placement of configuration $(n_0, n_1, \ldots, n_d)$). We pick an arbitrary point $p$ from the remaining points and consider all the possibilities of placing $p$ into a different box. For each branch where we place $p$ into a box $B$, we also remove all remaining points that are contained in the hyperplane determined by the points in the box $B$. The detailed algorithm is given in Fig. 2.

**Theorem 3.1.** Let $S$ be a set of $n$ points in the $m$-dimensional Euclidean space $\mathbb{R}^m$. The algorithm HyperplaneCover($S'; k, 0, \ldots, 0$) runs in time $O^*(((dk)!)/(d!)^k)) = O^*(k^{d-1}/c_{d-1}^k)$, where $c_{d-1} = \sqrt{2\pi d/e} > 1.3$, and correctly solves the $(d - 1)$-HYPERPLANE-COVER problem.

**Proof.** First consider the correctness of the algorithm. Note that each box contains at most $d$ points — this is because by step 4 of the algorithm, the new point $p$ from the set $S'$ is only added to boxes that contain fewer than $d$ points. Therefore, the hyperplane determined by the points in a box has its dimension bounded by $d - 1$. In particular, if $S' = \emptyset$, then the $k$ hyperplanes determined by the points in the $k$ boxes, respectively, all have their dimensions bounded by $d - 1$ and cover all points in $S'$. This proves the correctness of step 1. Inductively, if the algorithm produces a solution from one of its recursive executions in step 4, then the solution must be a collection of hyperplanes of dimensions bounded by $d - 1$ that cover all points in the set $S'$.

We must also prove that if the points in the set $S$ can be covered by $k$ hyperplanes of dimensions bounded by $d - 1$, then the algorithm HyperplaneCover($S'; k, 0, \ldots, 0$) must return a solution to the problem. For this, let $H_1, \ldots, H_k$ be $k$ hyperplanes of dimensions bounded by $d - 1$ such that all points in $S$ are contained in $\bigcup_{i=1}^{k} H_i$. Fix a partition of the points in $S$ into $k$ subsets $S_1, \ldots, S_k$ such that $S_i \subseteq H_i$ for all $i$ (note that the partition makes the subsets $S_1, \ldots, S_k$ pairwise disjoint, although a point in $S$ could belong to more than one of the hyperplanes).

Let $S', B_1, \ldots, B_k$ be subsets of the set $S$. We say that the collection $(S'; B_1, \ldots, B_k)$ is $(S_1, \ldots, S_k)$-consistent if the following conditions are all satisfied:

C1. $S', B_1, \ldots, B_k$ are pairwise disjoint;
C2. each $B_i$ contains at most $d$ points: $|B_i| \leq d$, and the dimension of $H[B_i]$ is $|B_i| - 1$;
C3. there is a permutation $\pi$ of $\{1, 2, \ldots, k\}$ such that $B_i \subseteq S_{\pi(i)}$ for all $i$;
C4. no point in $S'$ is in $H[B_i]$ for any $i$;
C5. every point in $S - S'$ is in $\bigcup_{i=1}^{k} H[B_i]$.

Each box placement can be given as a collection of $k$ subsets $B_1, \ldots, B_k$, where $B_i$ consists of the points in the $i$th box. We say that a subset $S'$ of $S$ plus the box placement is $(S_1, \ldots, S_k)$-consistent if the collection $(S'; B_1, \ldots, B_k)$ is $(S_1, \ldots, S_k)$-consistent.

**Claim.** Let $B = \{B_1, \ldots, B_k\}$ be a box placement and let $S' \neq \emptyset$ be a subset of $S$ such that $S' \cup B$ is $(S_1, \ldots, S_k)$-consistent. Then in the algorithm HyperplaneCover on input $S'$ plus $B$, one of the branches in step 4 must recursively call the algorithm on an input that is $(S_1, \ldots, S_k)$-consistent.

To prove the claim, first note that in this case the algorithm cannot stop at step 2: if $n_d = k$, then each box contains $d$ points. Then by condition C2, the hyperplane $H[B_i]$ has dimension $d - 1$. Since $B_i \subseteq S_{\pi(i)} \subseteq H_{\pi(i)}$, and the hyperplane $H_{\pi(i)}$ has its dimension bounded by $d - 1$, we must have $H_{\pi(i)} = H[B_i]$ for all $i$. Thus, if $S' \neq \emptyset$, then by condition C4, there are points in the set $S$ that are not contained in $\bigcup_{i=1}^{k} H_i$, contradicting the assumption that the hyperplanes $H_1, \ldots, H_k$ cover all the points in the set $S$.

Thus, on the input $S'$ and $B$, the algorithm must reach step 4 with a point $p$ in $S'$. Suppose $p \in S_j$, where we suppose that the mapping $\pi$ maps $B_i$ to $S_j$. By condition C4, $p \notin H[B_i]$. We must also have $|B_i| < d$: otherwise since $B_i \subseteq S_j \subseteq H_j$ and
$H_j$ has its dimension bounded by $d - 1$, by condition C2, $|B_i| = d$ would imply $H[B_i] = H_j$, contradicting our assumptions $p \not\in H[B_j]$ and $p \in S_j \subseteq H_j$. Therefore, one of the branches in step 4 will add $p$ to the box $B_i$ (i.e., the set $B_i$). It is easy to verify that this branch makes a recursive call to the problem on $(S_1, \ldots, S_k)$-consistent input. In particular, conditions C2–C3 are satisfied because of the above explanation and of the choice of the point $p$. Conditions C4–C5 are satisfied because we also remove all points in $S'$ that are contained in $H[B_i]$. This completes the proof of the claim.

When we start the algorithm HyperplaneCover($S', k, 0, \ldots, 0$) with the given set $S$ and $k$ empty boxes, the set $S$ plus the box placement is obviously $(S_1, \ldots, S_k)$-consistent. Therefore, by the above claim inductively, the execution of the algorithm always keeps at least one computational path $\mathcal{P}$ in its branching tree in which every node is associated with an $(S_1, \ldots, S_k)$-consistent input, which consists of a subset $S'$ of $S$ and a box placement. Each node on the path $\mathcal{P}$, when the subset $S'$ in the input is not empty, reduces the size of $S'$ and, by the above claim, must have a child in the path. Therefore, the path $\mathcal{P}$ must stop at a leaf $\mathcal{W}$ whose input has the subset $S' = \emptyset$. Note that the input on the leaf $\mathcal{W}$ is still $(S_1, \ldots, S_k)$-consistent. By definition, the box placement in the input of $\mathcal{W}$ must give $k$ hyperplanes of dimensions bounded by $d - 1$ that cover the set $S$.

This completes the proof of the correctness of the algorithm.

To derive the complexity of the algorithm, let $T'(n_0, n_1, \ldots, n_d)$ be the number of leaves in the branching tree $\mathcal{T}$ for the algorithm HyperplaneCover($S', n_0, n_1, \ldots, n_d$) on a given set $S'$ of points in $\mathbb{R}^m$ and a box placement of configuration $(n_0, n_1, \ldots, n_d)$. Then the function $T'(n_0, n_1, \ldots, n_d)$ satisfies the following recurrence relation (where for $0 \leq i \leq d - 1$,

$$T'(n_0, n_1, \ldots, n_d) = T'(n_0, n_1, \ldots, n_{i-1}, n_i - 1, n_{i+1} + 1, n_{i+2}, \ldots, n_d)$$

if $n_d = k$;

$$T'(n_0, n_1, \ldots, n_d) = \sum_{i=1}^{d-1} n_i T'(n_0, n_1, \ldots, n_d)$$

if $n_0 > 0$;

and

if $n_0 = 0$ and $n_d < k$.

The only difference between Formulas (2) and (1) is that in the last two cases, the equality "=" in (1) is replaced by "≤" in (2); this is because the algorithm HyperplaneCover may have an earlier exit because of step 1. It is trivial to verify by induction that

$$T'(n_0, n_1, \ldots, n_d) \leq T(n_0, n_1, \ldots, n_d)$$

for all $n_0, n_1, \ldots, n_d$. Thus, by Theorem 2.2, the algorithm HyperplaneCover($S, k, 0, \ldots, 0$) has its running time bounded by $O^*((((d+1)k)!)/(((d+1)!!)k^{!})) = O^*(k^{d-1}k^{1.35^d})$, where the first equality has used the Stirling approximation $r! \approx 2\pi r(r/e)^d$, and $c_d = \sqrt{2\pi d/e} > 1.3$ (here we assume $d \geq 2$ since otherwise the $(d - 1)$-HYPERPLANE-COVER problem is trivial). \hfill \Box

Replacing $d - 1$ by $d$ in Theorem 3.1, we conclude with the following complexity for the $d$-HYPERPLANE-COVER problem.

**Corollary 3.2.** For any integers $1 \leq d \leq m$, the $d$-HYPERPLANE-COVER problem in the Euclidean space $\mathbb{R}^m$ can be solved in time $O^*((((d+1)k)!)/(((d+1)!!)k^{!})) = O^*(k^{d-1}k^{1.35^d})$, where $c_d = \sqrt{2\pi d/e} > 1.3$.

We remark that the super-polynomial part in the complexity bound in Corollary 3.2 is independent of the dimension $m$ of the Euclidean space $\mathbb{R}^m$. In fact, it is not hard to verify that the algorithm HyperplaneCover solves the $d$-HYPERPLANE-COVER problem in time $O(nkd^d/mk^{d-1})$, where we assume an algorithm of time $O(d^2m)$ to test if a point in $\mathbb{R}^m$ is in a hyperplane of dimension bounded by $d$.

The standard HYPERPLANE-COVER problem in $\mathbb{R}^d$ (given a set $S$ of $n$ points in $\mathbb{R}^d$ that cover $S$, see [5]) is the $(d - 1)$-HYPERPLANE-COVER problem in $\mathbb{R}^d$. Therefore, our algorithm solves the standard HYPERPLANE-COVER problem in time $O^*(k^{d-1}k^{1.35^d})$, improving the previous best algorithm of time $O^*(k^{d+1})$ [5]. The LINE-COVER problem (given a set $S$ of $n$ points in the plane, decide if there are $k$ lines that cover $S$, see [2]) is the 1-HYPERPLANE-COVER problem in $\mathbb{R}^2$. Thus, our algorithm solves the LINE-COVER problem in time $O^*((2k)!)/(2^kk!) = O^*(k^{2}k^{1.35^d})$, improving the previous best algorithm of time $O^*(k^{2}/4.84^d)$ [2].

**Acknowledgements**

We would like to thank two anonymous referees whose comments and suggestions have helped in the presentation of the paper.

**References**


