Pinning down the Strong Wilber 1 Bound for Binary Search Trees

Parinya Chalermsook
Aalto University, Finland
chalermsook@gmail.com

Julia Chuzhoy
Toyota Technological Institute at Chicago, IL, USA
cjulia@ttic.edu

Thatchaphol Saranurak
Toyota Technological Institute at Chicago, IL, USA
saranurak@ttic.edu

Abstract

The dynamic optimality conjecture, postulating the existence of an $O(1)$-competitive online algorithm for binary search trees (BSTs), is among the most fundamental open problems in dynamic data structures. Despite extensive work and some notable progress, including, for example, the Tango Trees (Demaine et al., FOCS 2004), that give the best currently known $O(\log\log n)$-competitive algorithm, the conjecture remains widely open. One of the main hurdles towards settling the conjecture is that we currently do not have approximation algorithms achieving better than an $O(\log\log n)$-approximation, even in the offline setting. All known non-trivial algorithms for BST's so far rely on comparing the algorithm's cost with the so-called Wilber's first bound (WB-1). Therefore, establishing the worst-case relationship between this bound and the optimal solution cost appears crucial for further progress, and it is an interesting open question in its own right.

Our contribution is two-fold. First, we show that the gap between the WB-1 bound and the optimal solution value can be as large as $\Omega(\log\log n/\log\log\log n)$; in fact, we show that the gap holds even for several stronger variants of the bound. Second, we provide a simple algorithm, that, given an integer $D > 0$, obtains an $O(D)$-approximation in time $\exp\left(O\left(n^{1/2^{\Omega(D)}}\log n\right)\right)$. In particular, this yields a constant-factor approximation algorithm with sub-exponential running time. Moreover, we obtain a simpler and cleaner efficient $O(\log\log n)$-approximation algorithm that can be used in an online setting. Finally, we suggest a new bound, that we call the *Guillotine Bound*, that is stronger than WB-1, while maintaining its algorithm-friendly nature, that we hope will lead to better algorithms. All our results use the geometric interpretation of the problem, leading to cleaner and simpler analysis.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

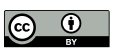
Keywords and phrases Binary search trees, Dynamic optimality, Wilber bounds

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.33

Category APPROX

Related Version A full version of the paper is available at `https://arxiv.org/abs/1912.02900`.

Funding *Parinya Chalermsook*: Supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 759557) and by Academy of Finland Research Fellows, under grant No. 310415.
Julia Chuzhoy: Supported in part by NSF grant CCF-1616584. Part of the work was done while the second author was a Weston visiting professor at the Department of Computer Science and Applied Mathematics, Weizmann Institute of Science.



1 Introduction

Binary search trees (BST's) are a fundamental data structure that has been extensively studied for many decades. Informally, suppose we are given as input an **online** access sequence $X = \{x_1, \ldots, x_m\}$ of keys from $\{1, \ldots, n\}$, and our goal is to maintain a binary search tree T over the set $\{1, \ldots, n\}$ of keys. The algorithm is allowed to modify the tree T after each access; the tree obtained after the ith access is denoted by T_{i+1}. Each such modification involves a sequence of *rotation* operations that transform the current tree T_i into a new tree T_{i+1}. The cost of the transformation is the total number of rotations performed plus the depth of the key x_i in the tree T_i. The total cost of the algorithm is the total cost of all transformations performed as the sequence X is processed. We denote by $\mathsf{OPT}(X)$ the smallest cost of any algorithm for maintaining a BST for the access sequence X, when the whole sequence X is known to the algorithm in advance.

Several algorithms for BST's, whose costs are guaranteed to be $O(m \log n)$ for any access sequence, such as AVL-trees [1] and red-black trees [2], are known since the 60's. Moreover, it is well known that there are length-m access sequences X on n keys, for which $\mathsf{OPT}(X) = \Omega(m \log n)$. However, such optimal worst-case guarantees are often unsatisfactory from both practical and theoretical perspectives, as one can often obtain better results for "structured" inputs. Arguably, a better notion of the algorithm's performance to consider is *instance optimality*, where the algorithm's performance is compared to the optimal cost $\mathsf{OPT}(X)$ for the specific input access sequence X. This notion is naturally captured by the algorithm's *competitive ratio*: we say that an algorithm for BST's is α-*competitive*, if, for every online input access sequence X, the cost of the algorithm's execution on X is at most $\alpha \cdot \mathsf{OPT}(X)$. Since for every length-m access sequence X, $\mathsf{OPT}(X) \geq m$, the above-mentioned algorithms that provide worst-case $O(m \log n)$-cost guarantees are also $O(\log n)$-competitive. However, there are many known important special cases, in which the value of the optimal solution is $O(m)$, and for which the existence of an $O(1)$-competitive algorithm would lead to a much better performance, including some interesting applications, such as, for example, adaptive sorting [23, 6, 19, 22, 13, 20, 12, 8, 7, 3, 5, 4]. A striking conjecture of Sleator and Tarjan [21] from 1985, called the *dynamic optimality conjecture*, asserts that the *Splay Trees* provide an $O(1)$-competitive algorithm for BST's. This conjecture has sparked a long line of research, but despite the continuing effort, and the seeming simplicity of BST's, it remains widely open. In a breakthrough result, Demaine et al. [10] proposed the Tango Trees algorithm, that achieves an $O(\log \log n)$-competitive ratio, and has remained the best known algorithm for the problem, for over 15 years. A natural avenue for overcoming this barrier is to first consider the "easier" task of designing (offline) approximation algorithms, whose approximation factor is below $O(\log \log n)$. Designing better approximation algorithms is often a precursor to obtaining better online algorithms, and it is a natural stepping stone towards this goal.

The main obstacle towards designing better algorithms, both in the online and the offline settings, is obtaining tight lower bounds on the value $\mathsf{OPT}(X)$, that can be used in algorithm design. If the input access sequence X has length m, and it contains n keys, then it is easy to see that $\mathsf{OPT}(X) \geq \Omega(m)$, and, by using any balanced BST's, such as AVL-trees, one can show that $\mathsf{OPT}(X) = O(m \log n)$. This trivially implies an $O(\log n)$-approximation for both offline and online settings. However, in order to obtain better approximation, these simple bounds do not seem sufficient. Wilber [25] proposed two new bounds, that we refer to as the first Wilber Bound (WB-1) and the second Wilber Bound (WB-2). He proved that, for every input sequence X, the values of both these bounds on X are at most $\mathsf{OPT}(X)$.

The breakthrough result of Demaine et al. [10], that gives an $O(\log\log n)$-competitive online algorithm, relies on the WB-1 bound. In particular, they show that the cost of the solution produced by their algorithm is within an $O(\log\log n)$-factor from the WB-1 bound on the given input sequence X, and hence from $\mathsf{OPT}(X)$. This in turn implies that, for every input sequence X, the value of the WB-1 bound is within an $O(\log\log n)$ factor from $\mathsf{OPT}(X)$. Follow-up work [24, 14] improved several aspects of Tango Trees, but it did not improve the approximation factor. Additional lower bounds on OPT, that subsume both the WB-1 and the WB-2 bounds, were suggested in [9, 11, 15], but unfortunately it is not clear how to exploit them in algorithm design. To this day, the only method we have for designing non-trivial online or offline approximation algorithms for BST's is by relying on the WB-1 bound, and this seems to be the most promising approach for obtaining better algorithms. In order to make further progress on both online and offline approximation algorithms for BST's, it therefore appears crucial that we better understand the relationship between the WB-1 bound and the optimal solution cost.

Informally, the WB-1 bound relies on recursive partitioning of the input key sequence, that can be represented by a partitioning tree. The standard WB-1 bound (that we refer to as the *weak* WB-1 bound) only considers a single such partitioning tree. It is well-known (see e.g. [10, 24, 16]), that the gap between $\mathsf{OPT}(X)$ and the weak WB-1 bound for an access sequence X may be as large as $\Omega(\log\log n)$. However, the "bad" access sequence X used to obtain this gap is highly dependent on the fixed partitioning tree T. It is then natural to consider a stronger variant of WB-1, that we refer to as *strong* WB-1 bound and denote by $\mathsf{WB}(X)$, that maximizes the weak WB-1 bound over all such partitioning trees. As suggested by Iacono [16], and by Kozma [17], this gives a promising approach for improving the $O(\log\log n)$-approximation factor.

In this paper, we show that, even for this strong variant of Wilber Bound, the gap between $\mathsf{OPT}(X)$ and $\mathsf{WB}(X)$ may be as large as $\Omega(\log\log n/\log\log\log n)$. This negative result extends to an even stronger variant of the Wilber Bound that we discuss below.

Our second set of results is algorithmic. We show an (offline) algorithm that, given an input sequence X and a positive integer D, obtains an $O(D)$-approximation, in time $\mathrm{poly}(m)\cdot\exp\left(n^{1/2^{\Omega(D)}}\log n\right)$. When D is constant, the algorithm obtains an $O(1)$-approximation in sub-exponential time. When D is $\Theta(\log\log n)$, it matches the best current efficient $O(\log\log n)$-approximation algorithm. In the latter case, we can also adapt the algorithm to the online setting, obtaining an $O(\log\log n)$-competitive online algorithm.

All our results use the geometric interpretation of the problem, introduced by Demaine et al. [9], leading to clean divide-and-conquer-style arguments that avoid, for example, the notion of pointers and rotations. We feel that this approach, in addition to providing a cleaner and simpler view of the problem, is more natural to work with in the context of approximation algorithms, and should be more amenable to the powerful geometric techniques in the field.

Independent Work. Independently from our work, Lecomte and Weinstein [18] showed that second Wilber Bound (WB-2) dominates the WB-1 bound, and moreover, they show an access sequence X for which the two bounds have a gap of $\Omega(\log\log n)$. In particular, their result implies that the gap between $\mathsf{WB}(X)$ and $\mathsf{OPT}(X)$ is $\Omega(\log\log n)$ for that access sequence. We note that the access sequence X that is used in our negative results also provides a gap of $\Omega(\log\log n/\log\log\log n)$ between the WB-2 and the WB-1 bounds, although we only realized this after hearing the statement of the results of [18]. Additionally, Lecomte and Weinstein show that the WB-2 bound is invariant under rotations, and use this to show that, when the WB-2 bound is constant, then the Independent Rectangle bound of [9] is linear.

We now provide a more detailed description of our results.

Our Results and Techniques

We use the geometric interpretation of the problem, introduced by Demaine et al. [9], that we refer to as the Min-Sat problem. Let P be any set of points in the plane. We say that two points $p, q \in P$ are *collinear* iff either their x-coordinates are equal, or their y-coordinates are equal. If p and q are non-collinear, then we let $\square_{p,q}$ be the smallest closed rectangle containing both p and q; notice that p and q must be diagonally opposite corners of this rectangle. We say that the pair (p, q) of points is *satisfied* in P iff there is some additional point $r \neq p, q$ in P that lies in $\square_{p,q}$ (the point may lie on the boundary of the rectangle). Lastly, we say that the set P of points is satisfied iff for every pair $p, q \in P$ of distinct points, either p and q are collinear, or they are satisfied in P.

In the Min-Sat problem, the input is a set P of points in the plane with integral x- and y-coordinates; we assume that all x-coordinates are between 1 and n, and all y-coordinates are between 1 and m and distinct from each other, and that $|P| = m$. The goal is to find a minimum-cardinality set Y of points, such that the set $P \cup Y$ of points is satisfied.

An access sequence X over keys $\{1, \ldots, n\}$ can be represented by a set P of points in the plane as follows: if a key x is accessed at time y, then add the point (x, y) to P. Demaine et al. [9] showed that, for every access sequence X, if we denote by P the corresponding set of points in the plane, then the value of the optimal solution to the Min-Sat problem on P is $\Theta(\mathsf{OPT}(X))$. They also showed that, in order to obtain an $O(\alpha)$-approximation algorithm for BST's, it is sufficient to obtain an α-approximation algorithm for the Min-Sat problem. In the online version of the Min-Sat problem, at every time step t, we discover the unique input point whose y-coordinate is t, and we need to decide which points with y-coordinate t to add to the solution. Demaine et al. [9] also showed that an α-competitive online algorithm for Min-Sat implies an $O(\alpha)$-competitive online algorithm for BST's. For convenience, we do not distinguish between the input access sequence X and the corresponding set of points in the plane, that we also denote by X.

Negative Results for WB-1. We say that an input access sequence X is a *permutation* if each key in $\{1, \ldots, n\}$ is accessed exactly once. Equivalently, in the geometric view, every column with an integral x-coordinate contains exactly one input point.

Informally, the WB-1 bound for an input sequence X is defined as follows. Let B be the bounding box containing all points of X, and consider any vertical line L drawn across B, that partitions it into two vertical strips, separating the points of X into two subsets X_1 and X_2. Assume that the points of X are ordered by their y-coordinates from smallest to largest. We say that a pair $(x, x') \in X$ of points *cross* the line L, iff x and x' are consecutive points of X, and they lie on different sides of L. Let $C(L)$ be the number of all pairs of points in X that cross L. We then continue this process recursively with X_1 and X_2, with the final value of the WB-1 bound being the sum of the two resulting bounds obtained for X_1 and X_2, and $C(L)$. This recursive partitioning process can be represented by a binary tree T that we call a *partitioning tree* (we note that the partitioning tree is not related to the BST tree that the BST algorithm maintains). Every vertex v of the partitioning tree is associated with a vertical strip $S(v)$, where for the root vertex r, $S(r) = B$. If the partitioning algorithm uses a vertical line L to partition the strip $S(v)$ into two sub-strips S_1 and S_2, then vertex v has two children, whose corresponding strips are S_1 and S_2. Note that every sequence of vertical lines used in the recursive partitioning procedure corresponds to a unique partitioning tree and vice versa. Given a set X of points and a partitioning tree T, we denote by $\mathsf{WB}_T(X)$ the WB-1 bound obtained for X while following the partitioning scheme defined by T. Wilber [25] showed that, for every partitioning tree T, $\mathsf{OPT}(X) \geq \Omega(\mathsf{WB}_T(X))$ holds. Moreover, Demaine et al. [10]

showed that, if T is a balanced tree, then $\mathsf{OPT}(X) \leq O(\log\log n) \cdot \mathsf{WB}_T(X)$. These two bounds are used to obtain the $O(\log\log n)$-competitive algorithm of [10]. We call this variant of WB-1, that is defined with respect to a fixed tree T, the *weak* WB-1 bound.

Unfortunately, it is well-known (see e.g. [10, 24, 16]), that the gap between $\mathsf{OPT}(X)$ and the weak WB-1 bound on an input X may be as large as $\Omega(\log\log n)$. In other words, for any fixed partitioning tree T, there exists an input X (that depends on T), with $\mathsf{WB}_T(X) \leq O(\mathsf{OPT}(X)/\log\log n)$. However, the construction of this "bad" input X depends on the fixed partitioning tree T. We consider a stronger variant of WB-1, that we refer to as *strong* WB-1 bound and denote by $\mathsf{WB}(X)$, that maximizes the weak WB-1 bound over all such partitioning trees, that is, $\mathsf{WB}(X) = \max_T\{\mathsf{WB}_T(X)\}$. Using this stronger bound as an alternative to weak WB-1 in order to obtain better approximation algorithms was suggested by Iacono [16], and by Kozma [17].

Our first result rules out this approach: we show that, even for the strong WB-1 bound, the gap between $\mathsf{WB}(X)$ and $\mathsf{OPT}(X)$ may be as large as $\Omega(\log\log n/\log\log\log n)$, even if the input X is a permutation.

▶ **Theorem 1.** *For every integer n', there is an integer $n \geq n'$, and an access sequence X on n keys with $|X| = n$, such that X is a permutation, $\mathsf{OPT}(X) \geq \Omega(n\log\log n)$, but $\mathsf{WB}(X) \leq O(n\log\log\log n)$. In other words, for every partitioning tree T, $\frac{\mathsf{OPT}(X)}{\mathsf{WB}_T(X)} \geq \Omega\left(\frac{\log\log n}{\log\log\log n}\right)$.*

We note that it is well known (see e.g. [5]), that any c-approximation algorithm for permutation input can be turned into an $O(c)$-approximation algorithm for any input sequence. However, the known instances that achieve an $\Omega(\log\log n)$-gap between the weak WB-1 bound and OPT are not permutations. Therefore, our result is the first to provide a super-constant gap between WB-1 and OPT for permutations, even for the case of weak WB-1.

Extension of WB-1. We consider several generalizations of the WB-1 bound that allow partitioning the plane both horizontally and vertically. We call the new bounds the *consistent Guillotine Bound* and the *Guillotine Bound*. Our negative result extends to the consistent Guillotine Bound but *not* to the Guillotine Bound. The Guillotine Bound seems to maintain the algorithm-friendly nature of WB-1, and in particular it naturally fits into the algorithmic framework that we propose. We hope that this bound can lead to improved algorithms, both in the offline and the online settings

Separating the Two Wilber Bounds. The sequence X given by Theorem 1 not only provides a separation between WB-1 and OPT, but it also provides a separation between the WB-1 bound and the WB-2 bound (also called the *funnel* bound). The latter can be defined in the geometric view as follows. Recall that, for a pair of points $x, y \in X$, $\square_{x,y}$ is the smallest closed rectangle containing both x and y. For a point x in the access sequence X, the *funnel* of x is the set of all points $y \in X$, for which $\square_{x,y}$ does not contain any point of $X \setminus \{x, y\}$, and $\mathrm{alt}(x)$ is the number of alterations between the left of x and the right of x in the funnel of x. The second Wilber Bound for sequence X is then defined as: $\mathsf{WB}^{(2)}(X) = |X| + \sum_{x\in X}\mathrm{alt}(x)$. We show that, for the sequence X given by Theorem 1, $\mathsf{WB}^{(2)}(X) \geq \Omega(n\log\log n)$ holds, and therefore $\mathsf{WB}^{(2)}(X)/\mathsf{WB}(X) \geq \Omega(\log\log n/\log\log\log n)$ for that sequence, implying that the gap between $\mathsf{WB}(X)$ and $\mathsf{WB}^{(2)}(X)$ may be as large as $\Omega(\log\log n/\log\log\log n)$. We note that we only realized that our results provide this stronger separation between the two Wilber bounds after hearing the statements of the results from the independent work of Lecomte and Weinstein [18] mentioned above.

Algorithmic Results. We provide new simple approximation algorithms for the problem, that rely on its geometric interpretation, namely the Min-Sat problem.

▶ **Theorem 2.** *There is an offline algorithm for Min-Sat, that, given any integral parameter $D \geq 1$, and an access sequence X to n keys of length m, produces a solution of cost at most $O(D \cdot OPT(X))$ and has running time at most* $\text{poly}(m) \cdot \exp\left(O\left(n^{1/2^{\Theta(D)}} \log n\right)\right)$. *For $D = O(\log \log n)$, the algorithm's running time is polynomial in n and m, and it can be adapted to the online setting, achieving an $O(\log \log n)$-competitive ratio.*

Our results show that the problem of obtaining a constant-factor approximation for Min-Sat cannot be NP-hard, unless NP $\subseteq$ SUBEXP, where SUBEXP $= \bigcap_{\epsilon>0} \text{DTime}[2^{n^\epsilon}]$. This, in turn, provides a positive evidence towards the dynamic optimality conjecture, as one natural avenue to disproving it is to show that obtaining a constant-factor approximation for BST's is NP-hard. Our results rule out this possibility, unless NP $\subseteq$ SUBEXP. While the $O(\log \log n)$-approximation factor achieved by our algorithm in time $\text{poly}(mn)$ is similar to that achieved by other known algorithms [10, 14, 24], this is the first algorithm that relies solely on the geometric formulation of the problem, which is arguably cleaner, simpler, and better suited for exploiting the rich toolkit of algorithmic techniques developed in the areas of online and approximation algorithms.

Organization. We start with preliminaries in Section 2. In Section 3, we state decomposition theorems which are useful for both of our negative and positive results. In Section 4, we provide the proof of Theorem 1, our main negative result. We discuss extensions of the Wilber Bound in Section 5. Lastly, we show our main positive result – the proof of Theorem 2 – in Section 6. Due to lack of space, many of the proofs are deferred to the full version.

2 Preliminaries

All our results only use the geometric interpretation of the problem, that we refer to as the Min-Sat problem. We include the formal definition of algorithms for BST's and formally state their equivalence to Min-Sat in the full version.

2.1 The Min-Sat Problem

For a point $p \in \mathbb{R}^2$ in the plane, we denote by $p.x$ and $p.y$ its x- and y-coordinates, respectively. Given any pair p, p' of points, we say that they are *collinear* if $p.x = p'.x$ or $p.y = p'.y$. If p and p' are not collinear, then we let $\square_{p,p'}$ be the smallest closed rectangle containing both p and p'; note that p and p' must be diagonally opposite corners of the rectangle.

▶ **Definition 3.** *We say that a non-collinear pair p, p' of points is* satisfied by a point p'' *if p'' is distinct from p and p' and $p'' \in \square_{p,p'}$ (where p'' may lie on the boundary of the rectangle). We say that a set S of points is* satisfied *iff for every non-collinear pair $p, p' \in S$ of points, there is some point $p'' \in S$ that satisfies this pair.*

We refer to horizontal and vertical lines as *rows* and *columns* respectively. For a collection of points X, the *active rows* of X are the rows that contain at least one point in X. We define the notion of *active columns* analogously. We denote by $r(X)$ and $c(X)$ the number of active rows and active columns of the point set X, respectively. We say that a point set X is a *semi-permutation* if every active row contains exactly one point of X. Note that, if X is a semi-permutation, then $c(X) \leq r(X)$. We say that X is a *permutation* if it is a

semi-permutation, and additionally, every active column contains exactly one point of X. Clearly, if X is a permutation, then $c(X) = r(X) = |X|$. We denote by B the smallest closed rectangle containing all points of X, and call B the *bounding box*.

We are now ready to define the Min-Sat problem. The input to the problem is a set X of points that is a semi-permutation, and the goal is to compute a minimum-cardinality set Y of points, such that $X \cup Y$ is satisfied. We say that a set Y of points is a *feasible solution* for X if $X \cup Y$ is satisfied. We denote by $\mathsf{OPT}(X)$ the minimum value $|Y|$ of any feasible solution Y for X.[1] In the online version of the Min-Sat problem, at every time step t, we discover the unique input point from X whose y-coordinate is t, and we need to decide which points with y-coordinate t to add to the solution Y. The Min-Sat problem is equivalent to the BST problem, in the following sense:

▶ **Theorem 4** ([9]). *Any efficient α-approximation algorithm for Min-Sat can be transformed into an efficient $O(\alpha)$-approximation algorithm for BST's, and similarly any online α-competitive algorithm for Min-Sat can be transformed into an online $O(\alpha)$-competitive algorithm for BST's.*

2.2 Basic Geometric Properties

The following observation is well known (see, e.g. Observation 2.1 from [9]).

▶ **Observation 5.** *Let Z be any satisfied point set. Then for every pair $p, q \in Z$ of distinct points, there is a point $r \in \square_{p,q} \setminus \{p, q\}$ such that $r.x = p.x$ or $r.y = p.y$.*

Collapsing Sets of Columns or Rows. Assume that we are given any set X of points, and any collection $\mathcal{C}$ of consecutive active columns for X. In order to collapse the set $\mathcal{C}$ of columns, we replace $\mathcal{C}$ with a single representative column C (for concreteness, we use the column of $\mathcal{C}$ with minimum x-coordinate). For every point $p \in X$ that lies on a column of $\mathcal{C}$, we replace p with a new point, lying on the column C, whose y-coordinate remains the same. Formally, we replace point p with point $(x, p.y)$, where x is the x-coordinate of the column C. We denote by $X_{|\mathcal{C}}$ the resulting new set of points. We can similarly define collapsing set of rows. The following useful observation is easy to verify.

▶ **Observation 6.** *Let S be any set of points, and let $\mathcal{C}$ be any collection of consecutive active columns (or rows) with respect to S. If S is a satisfied set of points, then so is $S_{|\mathcal{C}}$.*

Canonical Solutions. We say that a solution Y for input X is *canonical* iff every point $p \in Y$ lies on an active row and an active column of X. It is easy to see that *any* solution can be transformed into a canonical solution, without increasing its cost (see the full version of the paper for the proof).

▶ **Observation 7.** *There is an efficient algorithm, that, given an instance X of Min-Sat and any feasible solution Y for X, computes a feasible canonical solution $\hat{Y}$ for X with $|\hat{Y}| \leq |Y|$.*

2.3 Partitioning Trees

We now turn to define partitioning trees, that are central to both defining the WB-1 bound and to describing our algorithm.

[1] We remark that in the original paper that introduced this problem [9], the value of the solution is defined as $|X \cup Y|$, while our solution value is $|Y|$. It is easy to see that for any semi-permutation X and solution Y for X, $|Y| \geq \Omega(|X|)$ must hold, so the two definitions are equivalent to within factor 2.

Let X be the a set of points that is a semi-permutation. We can assume without loss of generality that every column with an integral x-coordinate between 1 and $c(X)$ inclusive contains at least one point of X. Let B be the bounding box of X. Assume that the set of active columns is $\{C_1, \ldots, C_a\}$, where $a = c(X)$, and that for all $1 \leq i \leq a$, the x-coordinate of column C_i is i. Let $\mathcal{L}$ be the set of all vertical lines with half-integral x-coordinates between $1 + 1/2$ and $a - 1/2$ (inclusive). Throughout, we refer to the vertical lines in $\mathcal{L}$ as *auxiliary columns*. Let σ be an arbitrary ordering of the lines of $\mathcal{L}$ and denote $\sigma = (L_1, L_2, \ldots, L_{a-1})$. We define a hierarchical partition of the bounding box B into vertical strips using σ, as follows. We perform $a - 1$ iterations. In the first iteration, we partition the bounding box B, using the line L_1, into two vertical strips, S_L and S_B. For $1 < i \leq a - 1$, in iteration i we consider the line L_i, and we let S be the unique vertical strip in the current partition that contains the line L_i. We then partition S into two vertical sub-strips by the line L_i. When the partitioning algorithm terminates, every vertical strip contains exactly one active column.

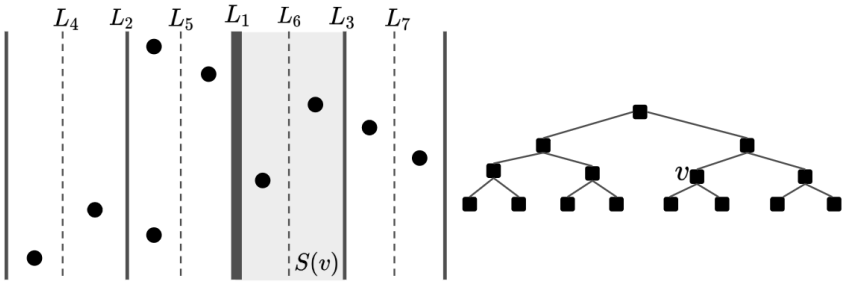


Figure 1 An Illustration of partitioning tree and the corresponding sequence $\sigma = (L_1, \ldots, L_7)$. Strip $S(v)$ corresponds to node v that owns line L_6.

This partitioning process can be naturally described by a binary tree $T = T(\sigma)$, that we call a *partitioning tree* associated with the ordering σ (see Figure 1). Each node $v \in V(T)$ is associated with a vertical strip $S(v)$ of the bounding box B. The strip $S(r)$ of the root vertex r of T is the bounding box B. For every inner vertex $v \in V(T)$, if $S = S(v)$ is the vertical strip associated with v, and if $L \in \mathcal{L}$ is the first line in σ that lies strictly in S, then line L partitions S into two sub-strips, that we denote by S_L and S_R. Vertex v then has two children, whose corresponding strips are S_L and S_R respectively. We say that v *owns* the line L, and we denote $L = L(v)$. For each leaf node v, the corresponding strip $S(v)$ contains exactly one active column of X, and v does not own any line of $\mathcal{L}$. For each vertex $v \in V(T)$, let $N(v) = |X \cap S(v)|$ be the number of points from X that lie in $S(v)$, and let width(v) be the width of the strip $S(v)$. Given a partition tree T for point set X, we refer to the vertical strips in $\{S(v)\}_{v \in T}$ as T-strips.

2.4 The WB-1 Bound

The WB-1 bound[2] is defined with respect to an ordering (or a permutation) σ of the auxiliary columns, or, equivalently, with respect to the partitioning tree $T(\sigma)$. It will be helpful to keep both these views in mind. In this paper, we will make a clear distinction between a weak variant of the WB-1 bound, as defined by Wilber himself in [25] and a strong variant, as mentioned in [16].

[2] Also called Interleaving bound [10], the first Wilber bound, "interleave lower bound" [25], or alternation bound [16]

Let X be a semi-permutation, and let $\mathcal{L}$ be the corresponding set of auxiliary columns. Consider an arbitrary fixed ordering σ of columns in $\mathcal{L}$ and its corresponding partition tree $T = T(\sigma)$. For each inner node $v \in V(T)$, consider the set $X' = X \cap S(v)$ of input points that lie in the strip $S(v)$, and let $L(v) \in \mathcal{L}$ be the line that v owns. We denote $X' = \{p_1, p_2, \ldots, p_k\}$, where the points are indexed in the increasing order of their y-coordinates; since X is a semi-permutation, no two points of X may have the same y-coordinate. For $1 \leq j < k$, we say that the ordered pair (p_j, p_{j+1}) of points form a *crossing* of $L(v)$ iff p_j, p_{j+1} lie on the opposite sides of the line $L(v)$. We let $\mathsf{cost}(v)$ be the total number of crossings of $L(v)$ by the points of $X \cap S(v)$. When $L = L(v)$, we also write $\mathsf{cost}(L)$ to denote $\mathsf{cost}(v)$. If v is a leaf vertex, then its cost is set to 0.

▶ **Definition 8** (WB-1 bound). *For any semi-permutation X, an ordering σ of the auxiliary columns in $\mathcal{L}$, and the corresponding partitioning tree $T = T_\sigma$, the (weak) WB-1 bound of X with respect to σ is: $\mathsf{WB}_\sigma(X) = \mathsf{WB}_T(X) = \sum_{v \in V(T)} \mathsf{cost}(v)$. The strong WB-1 bound of X is $\mathsf{WB}(X) = \max_\sigma \mathsf{WB}_\sigma(X)$, where the maximum is taken over all permutations σ of the lines in $\mathcal{L}$.*

It is well known that the WB-1 bound is a lower bound on the optimal solution cost:

▷ Claim 9. For any semi-permutation X, $\mathsf{WB}(X) \leq 2 \cdot \mathsf{OPT}(X)$.

The original proof of this fact is due to Wilber [25], which was later presented in the geometric view by Demaine et al. [9], via the notion of *independent rectangles*.

3 Geometric Decomposition Theorems

In this section, we develop several technical tools that will allow us to decompose a given instance into a number of sub-instances. We then analyze the optimal solution costs and the Wilber bound values for the resulting subinstances.

Split Instances. Consider a semi-permutation X and its partitioning tree T. Let $U \subseteq V(T)$ be a collection of vertices of the tree T, such that the strips $\{S(v)\}_{v \in U}$ partition the bounding box. In other words, every root-to-leaf path in T must contain exactly one vertex of U. We now define splitting an instance X via the set U of vertices of T.

▶ **Definition 10** (A Split). *A* split *of (X, T) at U is a collection of instances $\{X^c, \{X^s_v\}_{v \in U}\}$, defined as follows.*

- *For each vertex $v \in U$, instance X^s_v is called a* ***strip instance****, and it contains all points of X that lie in the interior of the strip $S(v)$.*
- *Instance X^c is called a* ***compressed instance****, and it is obtained from X by collapsing, for every vertex $v \in U$, all active columns in the strip $S(v)$ into a single column.*

We also partition the tree T into sub-trees that correspond to the new instances: for every vertex $v \in U$, we let T_v be the sub-tree of T rooted at v. Observe that T_v is a partitioning tree for instance X^s_v. The tree T^c is obtained from T by deleting from it, for all $v \in U$, all vertices of $V(T_v) \setminus \{v\}$. It is easy to verify that T^c is a valid partitioning tree for instance X^c. The following observation, whose proof appears in the full version of the paper, establishes several basic properties of a split. Recall that, given an instance X, $r(X)$ and $c(X)$ denote the number of active rows and active columns in X, respectively.

▶ **Observation 11.** *If X is a semi-permutation, then the following properties hold for any (X,T)-split at U:*

- $\sum_{v\in U} r(X^s_v) = r(X)$
- $\sum_{v\in U} c(X^s_v) = c(X)$
- $c(X^c) \leq |U|$
- $\sum_{v\in U} \mathsf{WB}_{T_v}(X^s_v) + \mathsf{WB}_{T^c}(X^c) = \mathsf{WB}_T(X)$.

The first property holds since X is a semi-permutation. In order to establish the last property, consider any vertex $x \in V(T)$, and let $T' \in \{T^c\} \cup \{T_v\}_{v\in U}$ be the new tree to which v belongs; if $x \in U$, then we set $T' = T_x$. It is easy to see that the cost of v in tree T' is the same as its cost in the tree T (recall that the cost of a leaf vertex is 0). The last property can be viewed as a "perfect decomposition" property of the weak WB-1 bound. We will show below an (approximate) decomposition property of strong WB-1 bound.

Splitting by Lines. We can also define the splitting with respect to any subset $\mathcal{L}' \subseteq \mathcal{L}$ of the auxiliary columns for X, analogously: Notice that the lines in $\mathcal{L}'$ partition the bounding box B into a collection of internally disjoint strips, that we denote by $\{S'_1, \ldots, S'_k\}$. We can then define the strip instances X^s_i as containing all vertices of $X \cap S_i$ for all $1 \leq i \leq k$, and the compressed instance X^c, that is obtained by collapsing, for each $1 \leq i \leq k$, all active columns that lie in strip S_i, into a single column. We also call these resulting instances *a split of X by $\mathcal{L}'$*.

We can also consider an arbitrary ordering σ of the lines in $\mathcal{L}$, such that the lines of $\mathcal{L}'$ appear at the beginning of σ, and let $U \subseteq V(T(\sigma))$ contain all vertices u for which the strip $S(u)$ is in $\{S_i\}_{1\leq i\leq k}$. If we perform a split of (X,T) at U, we obtain exactly the same strip instances $X^s_1, \ldots, X^s_k$, and the same compressed instance X^c.

Decomposition Theorem for OPT. The following theorem gives a crucial decomposition property of OPT. The theorem is used in our algorithm for Min-Sat, and its proof appears in the full version of the paper.

▶ **Theorem 12.** *Let X be a semi-permutation, let T be any partitioning tree for X, let $U \subseteq V(T)$ be a subset of vertices of T such that the strips in $\{S(v) \mid v \in U\}$ partition the bounding box, and let $\{X^c, \{X^s_v\}_{v\in U}\}$ be an (X,T)-split at U. Then:*

$$\sum_{v\in U} \mathit{OPT}(X^s_v) + \mathit{OPT}(X^c) \leq \mathit{OPT}(X).$$

Decomposition Theorem for the Strong WB-1 bound. We also prove, in the full version of the paper, the following theorem about the strong WB-1 bound, that we use several times in our negative result.

▶ **Theorem 13.** *Let X be a semi-permutation and let T be a partitioning tree for X. Let $U \subseteq V(T)$ be a set of vertices of T such that the strips in $\{S(v) \mid v \in U\}$ partition the bounding box. Let $\{X^c, \{X^s_v\}_{v\in U}\}$ be the split of (X,T) at U. Then:*

$$\mathsf{WB}(X) \leq 4\mathsf{WB}(X^c) + 8\sum_{v\in U} \mathsf{WB}(X^s_v) + O(|X|).$$

This result is somewhat surprising. One can think of the expression $\mathsf{WB}(X^c) + \sum_{v\in U} \mathsf{WB}(X^s_v)$ as a WB-1 bound obtained by first cutting along the lines that serve as boundaries of the strips $S(v)$ for $v \in U$, and then cutting the individual strips. However,

$\mathsf{WB}(X)$ is the maximum of $\mathsf{WB}_T(X)$ obtained over all trees T, including those that do not obey this partitioning order. The proofs of both Theorems 12 and 13 are given in the full version.

4 Separation of OPT and the Strong Wilber Bound

In this section we present our negative results, proving Theorem 1. We start by defining several basic tools used in our construction in Section 4.1. From Section 4.2 onward, we describe our construction and its analysis.

4.1 Basic Tools

Monotonically Increasing Sequence. We say that an input set X of points is *monotonically increasing* iff X is a permutation, and moreover for every pair $p, p' \in X$ of points, if $p.x < p'.x$, then $p.y < p'.y$ must hold. It is well known that the value of the optimal solution of monotonically increasing sequences is low, and we exploit this fact in our negative results.

▶ **Observation 14.** *If X is a monotonically increasing set of points, then $\mathsf{OPT}(X) \leq |X| - 1$.*

Bit Reversal Sequence (BRS). We use the geometric variant of BRS, which is more intuitive and easier to argue about. Let $\mathcal{R} \subseteq \mathbb{N}$ and $\mathcal{C} \subseteq \mathbb{N}$ be sets of integers (which are supposed to represent sets of active rows and columns.) The instance $\mathsf{BRS}(i, \mathcal{R}, \mathcal{C})$ is only defined when $|\mathcal{R}| = |\mathcal{C}| = 2^i$. It contains 2^i points, and it is a permutation, whose sets of active rows and columns are exactly $\mathcal{R}$ and $\mathcal{C}$ respectively; so $|\mathcal{R}| = |\mathcal{C}| = 2^i$. We define the instance recursively. The base of the recursion is instance $\mathsf{BRS}(0, \{C\}, \{R\})$, containing a single point at the intersection of row R and column C. Assume now that we have defined, for all $1 \leq i' \leq i$, and any sets $\mathcal{R}', \mathcal{C}'$ of $2^{i'}$ integers, the corresponding instance $\mathsf{BRS}(i, \mathcal{R}', \mathcal{C}')$. We define instance $\mathsf{BRS}(i+1, \mathcal{R}, \mathcal{C})$, where $|\mathcal{R}| = |\mathcal{C}| = 2^{i+1}$, as follows.

Consider the columns in $\mathcal{C}$ in their natural left-to-right order, and define $\mathcal{C}_{left}$ to be the first 2^i columns and $\mathcal{C}_{right} = \mathcal{C} \setminus \mathcal{C}_{left}$. Denote $\mathcal{R} = \{R_1, \ldots, R_{2^{i+1}}\}$, where the rows are indexed in their natural bottom to top order, and let $\mathcal{R}_{even} = \{R_2, R_4, \ldots, R_{2^{i+1}}\}$ and $\mathcal{R}_{odd} = \{R_1, R_3, \ldots, R_{2^{i+1}-1}\}$ be the sets of all even-indexed and all odd-indexed rows, respectively. Notice that $|\mathcal{C}_{left}| = |\mathcal{C}_{right}| = |\mathcal{R}_{even}| = |\mathcal{R}_{odd}| = 2^i$. The instance $\mathsf{BRS}(i+1, \mathcal{R}, \mathcal{C})$ is defined to be $\mathsf{BRS}(i, \mathcal{R}_{odd}, \mathcal{C}_{left}) \cup \mathsf{BRS}(i, \mathcal{R}_{even}, \mathcal{C}_{right})$.

For $n = 2^i$, we denote by $\mathsf{BRS}(n)$ the instance $\mathsf{BRS}(i, \mathcal{C}, \mathcal{R})$, where $\mathcal{C}$ contains all columns with integral x-coordinates from 1 to n, and $\mathcal{R}$ contains all rows with integral y-coordinates from 1 to n; see Figure 2 for an illustration.

It is well-known that, if X is a bit-reversal sequence on n points, then $\mathsf{OPT}(X) \geq \Omega(n \log n)$.

▷ Claim 15. Let $X = \mathsf{BRS}(i, \mathcal{C}, \mathcal{R})$, for any $i \geq 0$ and any sets $\mathcal{C}$ and $\mathcal{R}$ of columns and rows, respectively, with $|\mathcal{R}| = |\mathcal{C}| = 2^i$. Then $|X| = 2^i$, and $\mathsf{OPT}(X) \geq \frac{\mathsf{WB}(X)}{2} \geq \frac{|X|(\log |X|-2)+1}{2}$.

Next, we present two additional technical tools that we use in our construction.

Exponentially Spaced Columns. Recall that we defined the bit reversal instance $\mathsf{BRS}(\ell, \mathcal{R}, \mathcal{C})$, where $\mathcal{R}$ and $\mathcal{C}$ are sets of 2^ℓ rows and columns, respectively, that serve in the resulting instance as the sets of active rows and columns; the instance contains $n = 2^\ell$ points. In the Exponentially-Spaced BRS instance $\mathsf{ES\text{-}BRS}(\ell, \mathcal{R})$, we are still given a set $\mathcal{R}$ of 2^ℓ rows that will serve as active rows in the resulting instance, but we define the set

$\mathcal{C}$ of columns in a specific way. For an integer i, C_i be the column whose x-coordinate is i. We then let $\mathcal{C}$ contain, for each $0 \leq i < 2^\ell$, the column C_{2^i}. Denoting $N = 2^n = 2^{2^\ell}$, the x-coordinates of the columns in $\mathcal{C}$ are $\{1, 2, 4, 8, \ldots, N/2\}$. The instance is then defined to be $\mathsf{BRS}(\ell, \mathcal{R}, \mathcal{C})$ for this specific set $\mathcal{C}$ of columns. Notice that the instance contains $n = \log N = 2^\ell$ input points.

It is easy to see that any point set $X = \mathsf{ES\text{-}BRS}(\ell, \mathcal{R})$ satisfies $\mathsf{OPT}(X) = \Omega(n \log n)$. We remark that this idea of exponentially spaced columns is inspired by the instance used by Iacono [16] to prove a gap between the weak WB-1 bound and $\mathsf{OPT}(X)$. However, Iacono's instance is tailored to specific partitioning tree T, and it is clear that there is another partitioning tree T' with $\mathsf{OPT}(X) = \Theta(WB_{T'}(X))$. Therefore, this instance does not give a separation result for the strong WB-1 bound, and in fact it does not provide negative results for the weak WB-1 bound when the input point set is a permutation.

Cyclic Shift of Columns. Suppose we are given a point set X, and let $\mathcal{C}' = \{C_0, \ldots, C_{N-1}\}$ be any set of columns indexed in their natural left-to-right order, such that all points of X lie on columns of $\mathcal{C}$ (but some columns may contain no points of X). Let $0 \leq s < N$ be any integer. We denote by X^s a cyclic shift of X by s units, obtained as follows. For every point $p \in X$, we add a new point p^s to X^s, whose y-coordinate is the same as that of p, and whose x-coordinate is $p.x + s \mod N$. In other words, we shift the point p by s steps to the right in a circular manner. Equivalently, we move the last s columns of $\mathcal{C}'$ to the beginning of the instance. The following claim, whose proof appears in the full version of the paper, shows that the value of the optimal solution does not decrease significantly in the shifted instance.

▷ Claim 16. Let X be any point set that is a semi-permutation. Let $0 \leq s < N$ be a shift value, and let $X' = X^s$ be the instance obtained from X by a cyclic shift of its points by s units to the right. Then $\mathsf{OPT}(X') \geq \mathsf{OPT}(X) - |X|$.

4.2 Construction of the Bad Instance

We construct two instances: instance $\hat{X}$ on N^* points, that is a semi-permutation (but is somewhat easier to analyze), and instance X^* in N^* points, which is a permutation; the analysis of instance X^* heavily relies on the analysis of instance $\hat{X}$. We will show that the optimal solution value of both instances is $\Omega(N^* \log\log N^*)$, but the cost of the Wilber Bound is at most $O(N^* \log\log\log N^*)$. Our construction uses the following three parameters. We let $\ell \geq 1$ be an integer, and we set $n = 2^\ell$ and $N = 2^n$.

First Instance. We now construct our first final instance $\hat{X}$, which is a semi-permutation containing N columns. Intuitively, we create N instances $X^0, X^1, \ldots, X^{N-1}$, where instance X^s is an exponentially-spaced BRS instance that is shifted by s units. We then stack these instances on top of one another in this order.

Formally, for all $0 \leq j \leq N-1$, we define a set $\mathcal{R}_j$ of n consecutive rows with integral coordinates, such that the rows of $\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_{N-1}$ appear in this bottom-to-top order. Specifically, set $\mathcal{R}_j$ contains all rows whose y-coordinates are in $\{jn+1, jn+2, \ldots, (j+1)n\}$.

For every integer $0 \leq s \leq N-1$, we define a set of points X^s, which is a cyclic shift of instance $\mathsf{ES\text{-}BRS}(\ell, \mathcal{R}_s)$ by s units. Recall that $|X^s| = 2^\ell = n$ and that the points in X^s appear on the rows in $\mathcal{R}_s$ and a set $\mathcal{C}_s$ of columns, whose x-coordinates are in $\{\left(2^j + s\right) \mod N : 0 \leq j < n\}$. We then let our final instance be $\hat{X} = \bigcup_{s=0}^{N-1} X^s$. From now on, we denote $N^* = |\hat{X}|$. Recall that $|N^*| = N \cdot n = N \log N$.

Observe that the number of active columns in $\hat{X}$ is N. Since the instance is symmetric and contains $N^* = N \log N$ points, every column contains exactly $\log N$ points. Each row contains exactly one point, so $\hat{X}$ is a semi-permutation. (See Figure 2 for an illustration).

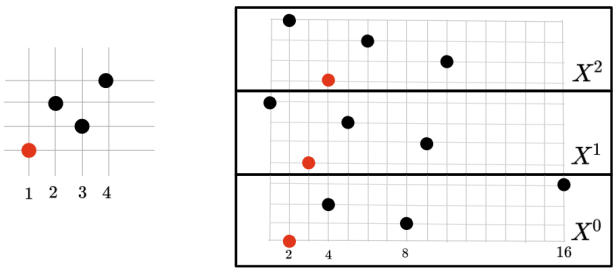


Figure 2 An illustration of our construction. The figure on the left shows the instance $\mathsf{BRS}(2, \{1,2,3,4\}, \{1,2,3,4\})$. The figure on the right combines three copies X^0, X^1, X^2 of the corresponding exponentially-spaced instance, with horizontal shifts of 0, 1, and 2, respectively. The red points show how copies of the same point in different sub-instances.

Lastly, we need the following bound on the value of the optimal solution of instance $\hat{X}$.

▶ **Observation 17.** $OPT(\hat{X}) = \Omega(N^* \log\log N^*)$

Proof. From Claims 15 and 16, for each $0 \leq s \leq N-1$, each sub-instance X^s has $\mathsf{OPT}(X^s) \geq \Omega(n \log n) = \Omega(\log N \log\log N)$. Therefore, $\mathsf{OPT}(\hat{X}) \geq \sum_{s=0}^{N-1} \mathsf{OPT}(X^s) = \Omega(N \log N \log\log N) = \Omega(N^* \log\log N^*)$ (we have used the fact that $N^* = N \log N$). ◀

Second Instance. We now construct our second and final instance, X^*, that is a permutation. In order to do so, we start with the instance $\hat{X}$, and, for every active column C of $\hat{X}$, we create $n = \log N$ new columns (that we view as copies of C), $C^1, \ldots, C^{\log N}$, which replace the column C. We denote this set of columns by $\mathcal{B}(C)$, and we refer it as the *block of columns representing* C. Recall that the original column C contains $\log N$ input points of $\hat{X}$. We place each such input point on a distinct column of $\mathcal{B}(C)$, so that the points form a monotonically increasing sequence (see the definition in Section 4.1). This completes the definition of the final instance X^*. We obtain the following immediate bound on the optimal solution cost of instance X^*.

▷ Claim 18. $\mathsf{OPT}(X^*) \geq \mathsf{OPT}(\hat{X}) = \Omega(N^* \log\log N^*)$.

4.3 Upper Bound for $\mathsf{WB}(\hat{X})$

In this section we prove the following theorem.

▶ **Theorem 19.** $\mathsf{WB}(\hat{X}) \leq O(N^* \log\log\log N^*)$.

In order to prove the theorem, consider again the instance $\hat{X}$. Recall that it consists of N instances $X^0, X^1, \ldots, X^{N-1}$ that are stacked on top of each other vertically in this order. We rename these instances as $X_1, X_2, \ldots, X_N$, so X_j is exactly $\mathsf{ES\text{-}BRS}(\log N)$, that is shifted by $(j-1)$ units to the right. Recall that $|\hat{X}| = N^* = N \log N$, and each instance X_s contains exactly $\log N$ points. We denote by $\mathcal{C}$ the set of N columns, whose x-coordinates are $1, 2, \ldots, N$. All points of $\hat{X}$ lie on the columns of $\mathcal{C}$. For convenience, for $1 \leq j \leq N$, we denote by C_j the column of $\mathcal{C}$ whose x-coordinate is j.

Let σ be any ordering of the auxiliary columns in $\mathcal{L}$, and let $T = T_\sigma$ be the corresponding partitioning tree. It is enough to show that, for any such ordering σ, the value of $\mathsf{WB}_\sigma(\hat{X})$ is bounded by $O(N^* \log\log\log N^*)$. Recall that $WB_\sigma(\hat{X})$ is the sum, over all vertices $v \in V(T)$, of $\mathsf{cost}(v)$. The value of $\mathsf{cost}(v)$ is defined as follows. If v is a leaf vertex, then $\mathsf{cost}(v) = 0$. Otherwise, let $L = L(v)$ be the line of $\mathcal{L}$ that v owns. Index the points in $X \cap S(v)$ by $q_1, \ldots, q_z$ in their bottom-to-top order. A consecutive pair (q_j, q_{j+1}) of points is a *crossing* iff they lie on different sides of $L(v)$. We distinguish between the two types of crossings that contribute towards $\mathsf{cost}(v)$. We say that the crossing (q_j, q_{j+1}) is of *type*-1 if both q_j and q_{j+1} belong to the same shifted instance X_s for some $0 \leq s \leq N-1$. Otherwise, they are of type-2. Note that, if (q_j, q_{j+1}) is a crossing of type 2, with $q_j \in X_s$ and $q_{j+1} \in X_{s'}$, then s, s' are not necessarily consecutive integers, as it is possible that for some indices s'', $X_{s''}$ has no points that lie in the strip $S(v)$. We now let $\mathsf{cost}_1(v)$ be the total number of type-1 crossings of $L(v)$, and $\mathsf{cost}_2(v)$ the total number of type-2 crossings. Note that $\mathsf{cost}(v) = \mathsf{cost}_1(v) + \mathsf{cost}_2(v)$. We also define $\mathsf{cost}_1(\sigma) = \sum_{v \in V(T)} \mathsf{cost}_1(v)$ and $\mathsf{cost}_2(\sigma) = \sum_{v \in V(T)} \mathsf{cost}_2(v)$. Clearly, $\mathsf{WB}_\sigma(\hat{X}) = \mathsf{cost}_1(\sigma) + \mathsf{cost}_2(\sigma)$. In the full version of the paper, we prove the following two theorems:

▶ **Theorem 20.** *For every ordering σ of the auxiliary columns in $\mathcal{L}$, $\mathsf{cost}_1(\sigma) \leq O(N^* \log\log\log N^*)$.*

▶ **Theorem 21.** *For every vertex $v \in V(T)$, $\mathsf{cost}_2(v) \leq O(\log N) + O(\mathsf{cost}_1(v))$.*

Notice that from the latter theorem, we get that $\mathsf{cost}_2(\sigma) \leq O(\mathsf{cost}_1(\sigma)) + O(|V(T)| \cdot \log N) = O(N^* \log\log\log N^*) + O(N \log N) = O(N^* \log\log\log N^*)$. Combining the two theorems together completes the proof of Theorem 19.

4.4 Upper Bound for $\mathsf{WB}(X^*)$

In this section we show that $\mathsf{WB}(X^*) = O(N^* \log\log\log N^*)$, completing the proof of Theorem 1. Recall that instance X^* is obtained from instance $\hat{X}$ by replacing every active column C of X^* with a block $\mathcal{B}(C)$ of columns, and then placing the points of C on the columns of $\mathcal{B}(C)$ so that they form a monotone increasing sequence, while preserving their y-coordinates. The resulting collection of all blocks $\mathcal{B}(C)$ partitions the set of all active columns of X^*. We denote this set of blocks by $\mathcal{B}_1, \ldots, \mathcal{B}_N$. The idea is to use Theorem 13 in order to bound $\mathsf{WB}(X^*)$.

Consider a set of lines $\mathcal{L}'$ (with half-integral x-coordinates) that partition the bounding box B into strips, where the ith strip contains the block $\mathcal{B}_i$ of columns, so $|\mathcal{L}'| = (N-1)$. We consider a split of instance X^* by $\mathcal{L}'$: This gives us a collection of strip instances $\{X_i^s\}_{1 \leq i \leq N}$ and the compressed instance X^c. Notice that the compressed instance is precisely $\hat{X}$, and each strip instance X_i^s is a monotone increasing point set.

Since each strip instance X_i^s is monotonously increasing, from Observation 14 and Claim 9, for all i, $\mathsf{WB}(X_i^s) \leq O(\mathsf{OPT}(X_i^s)) \leq O(|X_i^s|)$. From Theorem 13, we then get that: $\mathsf{WB}(X^*) \leq 4\mathsf{WB}(\hat{X}) + 8\sum_i \mathsf{WB}(X_i^s) + O(|X^*|) \leq 4\mathsf{WB}(\hat{X}) + O(|X^*|) \leq O(N^* \log\log\log N^*)$.

5 Guillotine Bounds

In this section we consider an extension of the Wilber bound which we call the Guillotine bound. The *Guillotine bound* $\mathsf{GB}(X)$ extends $\mathsf{WB}(X)$ by allowing both vertical and horizontal partitioning lines. Specifically, given the bounding box B, we let L be any vertical or horizontal line crossing B, that separates X into two subsets X_1 and X_2. We define the

number of crossings of L exactly as before, and then recurse on both sides of L as before. This partitioning scheme can be represented by a binary tree T, where every vertex of the tree is associated with a rectangular region of the plane. We denote the resulting bound obtained by using the partitioning tree T by $\mathsf{GB}_T(X)$, and we define $\mathsf{GB}(X) = \max_T \mathsf{GB}_T(X)$. We show that GB is a lower bound on the optimal solution cost in the following lemma, whose proof is deferred to the full version.

▶ **Lemma 22.** *For any point set X that is a permutation, $\mathsf{GB}(X) \leq 2\mathit{OPT}(X)$.*

The *Consistent Guillotine bound* restricts the Guillotine bound by maximizing only over partitioning schemes that are "consistent" in the following sense: suppose that the current partition of the bounding box B, that we have obtained using previous partitioning lines, is a collection $\{R_1, \ldots, R_k\}$ of rectangular regions. We need to choose a vertical or a horizontal line L that spans the whole bounding box B, that is, L intersects the boundary of B in two points. Once line L is chosen, for **every** rectangular region R_i that intersects L, we must partition R_i into two sub-regions using the line L, and then count the number of consecutive pairs of points in $X \cap R_i$ that cross the line L. In other words, we must partition all rectangles $R_1, \ldots, R_k$ consistently with respect to the line L. In contrast, in the Guillotine bound, we are allowed to partition each area R_i independently. From the definitions, the value of the Guillotine bound $\mathsf{GB}(X)$ is always at least as large as the value of the Consistent Guillotine bound, denoted by $\mathsf{cGB}(X)$, on any input sequence X, which is at least as large as $\mathsf{WB}(X)$. We generalize our negative result to the Consistent Guillotine bound in the following theorem, whose proof appears in the full version of the paper.

▶ **Theorem 23.** *For every integer n', there is an integer $n \geq n'$, and a set X of points that is a permutation with $|X| = n$, such that $\mathit{OPT}(X) \geq \Omega(n \log\log n)$ but $\mathsf{cGB}(X) \leq O(n \log\log\log n)$.*

Our negative results do not extend to the general GB bound, while our divide-and-conquer framework can naturally be adapted to work with GB. We leave open an interesting question of establishing the worst-case gap between the value of OPT and that of the Guillotine bound, and we hope that combining the Guillotine bound with our algorithmic framework will lead to better online and offline approximation algorithms.

6 The Algorithms

In this section we provide the high level intuition for the proof of Theorem 2. A more detailed description appears in the Appendix. Both the polynomial time and the sub-exponential time algorithms follow the same framework. We start with a high-level overview of this framework. For simplicity, assume that the number of active columns in the input instance X is an integral power of 2. The key idea is to decompose the input instance into smaller sub-instances, using the split instances defined in Section 3. We solve the resulting instances recursively and then combine the resulting solutions.

Suppose we are given an input point set X that is a semi-permutation, with $|X| = m$, such that the number of active columns is n. We consider a *balanced* partitioning tree T, where for every vertex $v \in V(T)$, the line $L(v)$ that v owns splits the strip $S(v)$ in the middle, with respect to the active columns that are contained in $S(v)$. Therefore, the height of the partitioning tree is $\log n$.

Consider now the set U of vertices of T that lie in the middle layer of T. We consider the split of (X, T) at U, obtaining a new collection of instances $(X^c, \{X^s_i\})_{i=1}^k$ where $k = \Theta(\sqrt{n})$. Note that each resulting strip instance X^s_i contains $\Theta(\sqrt{n})$ active columns, and so does the compressed instance X^c.

We recursively solve each such instance and then combine the resulting solutions. The key to the algorithm and its analysis is to show that there is a collection Z of $O(|X|)$ points, such that, if we are given any solution Y^c to instance X^c, and, for all $1 \leq i \leq k$, any solution Y_i to instance X_i^s, then $Z \cup Y^c \cup \left(\bigcup_{i=1}^{N} Y_i\right)$ is a feasible solution to instance X. We also show that the total number of input points that appear in all instances that participate in the same recursive level is bounded by $O(\mathsf{OPT}(X))$. This ensures that in every recursive level we add at most $O(\mathsf{OPT}(X))$ points to the solution, and the total solution cost is at most $O(\mathsf{OPT}(X))$ times the number of the recursive levels, which is bounded by $O(\log \log n)$.

In order to obtain the sub-exponential time algorithm, we restrict the recursion to D levels, and then solve each resulting instance directly in time $r(X)c(X)^{O(c(X))}$. This approach gives an $O(D)$-approximation algorithm with running time at most $\mathrm{poly}(m) \cdot \exp\left(n^{1/2^{\Omega(D)}} \log n\right)$ as desired. A more detailed description of the algorithm appears in the Appendix.

References

1. G. M. Adelson-Velskiĭ and E. M. Landis. An algorithm for organization of information. *Dokl. Akad. Nauk SSSR*, 146:263–266, 1962.
2. Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Inf.*, 1:290–306, 1972. doi:10.1007/BF00289509.
3. Prosenjit Bose, Karim Douïeb, John Iacono, and Stefan Langerman. The power and limitations of static binary search trees with lazy finger. In *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, pages 181–192, 2014. doi:10.1007/978-3-319-13075-0_15.
4. P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, and T. Saranurak. Greedy is an almost optimal deque. *WADS*, 2015.
5. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 410–423, 2015.
6. R. Chaudhuri and H. Höft. Splaying a search tree in preorder takes linear time. *SIGACT News*, 24(2):88–93, April 1993. doi:10.1145/156063.156067.
7. R. Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000. doi:10.1137/S009753979732699X.
8. Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. part i: Splay sorting log n-block sequences. *SIAM J. Comput.*, 30(1):1–43, April 2000. doi:10.1137/S0097539797326988.
9. Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *SODA 2009*, pages 496–505, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496825.
10. Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality - almost. *SIAM J. Comput.*, 37(1):240–251, 2007. Announced at FOCS'04. doi:10.1137/S0097539705447347.
11. Jonathan Derryberry, Daniel Dominic Sleator, and Chengwen Chris Wang. A lower bound framework for binary search trees with rotations, 2005.
12. Jonathan C. Derryberry and Daniel D. Sleator. Skip-splay: Toward achieving the unified bound in the BST model. In Frank Dehne, Marina Gavrilova, Jörg-Rüdiger Sack, and CsabaD. Toth, editors, *Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer Science*, pages 194–205. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-03367-4_18.
13. Amr Elmasry. On the sequential access theorem and deque conjecture for splay trees. *Theoretical Computer Science*, 314(3):459–466, 2004. doi:10.1016/j.tcs.2004.01.019.

14 George F. Georgakopoulos. Chain-splay trees, or, how to achieve and prove loglogn-competitiveness by splaying. *Inf. Process. Lett.*, 106(1):37–43, 2008. doi:10.1016/j.ipl.2007.10.001.

15 Dion Harmon. *New Bounds on Optimal Binary Search Trees*. PhD thesis, Massachusetts Institute of Technology, 2006.

16 John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40273-9_16.

17 László Kozma. *Binary search trees, rectangles and patterns*. PhD thesis, Saarland University, Saarbrücken, Germany, 2016. URL: http://scidok.sulb.uni-saarland.de/volltexte/2016/6646/.

18 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber's bounds for dynamic optimality. *arXiv preprint*, 2019. arXiv:1912.02858.

19 Joan M. Lucas. On the competitiveness of splay trees: Relations to the union-find problem. *On-line Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:95–124, 1991.

20 Seth Pettie. Splay trees, Davenport-Schinzel sequences, and the deque conjecture. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 1115–1124, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1347082.1347204.

21 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985. Announced at STOC'83. doi:10.1145/3828.3835.

22 Rajamani Sundar. On the deque conjecture for the splay algorithm. *Combinatorica*, 12(1):95–124, 1992. doi:10.1007/BF01191208.

23 Robert Endre Tarjan. Sequential access in splay trees takes linear time. *Combinatorica*, 5(4):367–378, 1985. doi:10.1007/BF02579253.

24 Chengwen C. Wang, Jonathan C. Derryberry, and Daniel D. Sleator. O(log log n)-competitive dynamic binary search trees. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 374–383, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109600.

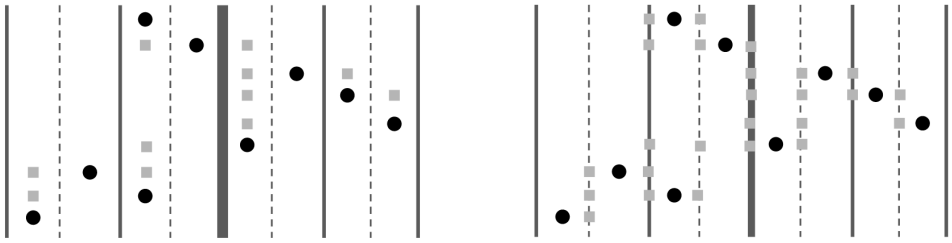
25 R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on Computing*, 18(1):56–67, 1989. Announced at FOCS'86. doi:10.1137/0218004.

A A Detailed Description of the Algorithms

In this section we provide additional details for the proof of Theorem 2. Due to lack of space, some of the proofs are deferred to the full version.

A.1 Special Solutions

Our algorithm will produce feasible solutions of a special form, that we call *special solutions*. Recall that, given a semi-permutation point set X, the auxiliary columns for X are a set $\mathcal{L}$ of vertical lines with half-integral coordinates. We say that a solution Y for X is *special* iff every point of Y lies on an row that is active for X, and on a column of $\mathcal{L}$. In particular, special solutions are by definition non-canonical (see Figure 3 for an illustration). The main advantage of the special solutions is that they allow us to easily use the divide-and-conquer approach. We use the following observation, whose proof appears in the full version of the paper.



(a) Canonical Solution. **(b)** Special Solution.

Figure 3 Canonical and T-special solutions of X. The input points are shown as circles; the points that belong to the solution Y are shown as squares.

▶ **Observation 24.** *There is an algorithm, that, given a set X of points that is a semi-permutation, and a canonical solution Y for X, computes a special solution Y' for X, such that $|Y'| \leq 2|X| + 2|Y|$. The running time of the algorithm is $O(|X| + |Y|)$.*

If σ is any ordering of the auxiliary columns in $\mathcal{L}$, and $T = T_\sigma$ is the corresponding partitioning tree, then any point set Y that is a special solution for X is also called a *T-special solution* (although the notion of the solution Y being special does not depend on the tree T, this notion will be useful for us later; in particular, a convenient way of thinking of a T-special solution is that every point of Y must lie on an active row of X, and on a column that serves as a boundary for some strip $S(v)$, where $v \in V(T)$.)

A.2 Redundant Points and Reduced Point Sets

Consider a semi-permutation X, that we think of as a potential input to the Min-Sat problem. We denote $X = \{p_1, \ldots, p_m\}$, where the points are indexed in their natural bottom-to-top order, so $(p_1).y < (p_2).y < \ldots < (p_m).y$. A point p_i is said to be *redundant*, iff $(p_i).x = (p_{i+1}).x = (p_{i-1}).x$. We say that a semi-permutation X is in the *reduced form* if there are no redundant points in X; in other words, if p_{i-1}, p_i, p_{i+1} are three points lying on three consecutive active rows, then their x-coordinates are not all equal. We use the following observation and lemma, whose proofs appear in the full version of the paper.

▶ **Observation 25.** *Let X be a semi-permutation, and let $X' \subseteq X$ be any point set, that is obtained from X by repeatedly removing redundant points. Then $\mathsf{OPT}(X') \leq \mathsf{OPT}(X)$.*

▶ **Lemma 26.** *Let X be a semi-permutation, and let $X' \subseteq X$ be any point set, that is obtained from X by repeatedly removing redundant points. Let Y be any feasible solution for X' such that every point of Y lies on a row that is active for X'. Then Y is also a feasible solution for X.*

From Lemma 26, whenever we need to solve the Min-Sat problem on an instance X, it is sufficient to solve it on a sub-instance, obtained by iteratively removing redundant points from X. We obtain the following immediate corollary of Lemma 26.

▶ **Corollary 27.** *Let X be a semi-permutation, and let $X' \subseteq X$ be any point set, that is obtained from X by repeatedly removing redundant points. Let Y be any special feasible solution for X'. Then Y' is also a special feasible solution for X.*

Lastly, we need the following lemma, which is a simple application of the Wilber bound.

▶ **Lemma 28.** *Let X be a point set that is a semi-permutation in reduced form. Then $\mathsf{OPT}(X) \geq |X|/4 - 1$.*

A.3 The Algorithm Description

Suppose we are given an input set X of points that is a semi-permutation. Let T be any partitioning tree for X. We say that T is a *balanced* partitioning tree for X iff for every non-leaf vertex $v \in V(T)$ the following holds. Let v' and v'' be the children of v in the tree T. Let X' be the set of all input points lying in strip $S(v)$, and let X'', X''' be defined similarly for $S(v')$ and $S(v'')$. Let c be the number of active columns in instance X', and let c' and c'' be defined similarly for X'' and X'''. Then we require that $c', c'' \leq \lceil c/2 \rceil$.

Given a partitioning tree T, we denote by Λ_i the set of all vertices of T that lie in the ith *layer* of T – that is, the vertices whose distance from the root of T is i (so the root belongs to Λ_0). The *height* of the tree T, denoted by $\text{height}(T)$, is the largest index i such that $\Lambda_i \neq \emptyset$. If the height of the tree T is h, then we call the set $\Lambda_{\lceil h/2 \rceil}$ of vertices the *middle layer* of T. Notice that, if T is a balanced partitioning tree for input X, then its height is at most $2 \log c(X)$.

Our algorithm takes as input a set X of points that is a semi-permutation, a balanced partition tree T for X, and an integral parameter $\rho > 0$.

Intuitively, the algorithm uses the splitting operation to partition the instance X into subinstances that are then solved recursively, until it obtains a collection of instances whose corresponding partitioning trees have height at most ρ. We then either employ dynamic programming, or use a trivial $O(\log c(X))$-approximation algorithm. The algorithm returns a special feasible solution for the instance. Recall that the height of the tree T is bounded by $2 \log c(X) \leq 2 \log n$. The following two theorems will be used as the recursion basis.

▶ **Theorem 29.** *There is an algorithm called* LeafBST-1 *that, given a semi-permutation instance X of* Min-Sat *in reduced form, and a partitioning tree T for it, produces a feasible T-special solution for X of cost at most $2|X| + 2\mathsf{OPT}(X)$, in time $|X|^{O(1)} \cdot c(X)^{O(c(X))}$.*

▶ **Theorem 30.** *There is an algorithm called* LeafBST-2 *that, given a semi-permutation instance X of* Min-Sat *in reduced form, and a partitioning tree T for it, produces a feasible T-special solution of cost at most $2|X| \cdot \text{height}(T)$, in time $\text{poly}(|X|)$.*

The proofs of both theorems are deferred to the full version of the paper. We now provide a schematic description of our algorithm. Depending on the guarantees that we would like to achieve, whenever the algorithm calls procedure LeafBST, it will call either procedure LeafBST-1 or procedure LeafBST-2; we specify this later.

Algorithm 1 RecursiveBST(X, T, ρ).

1. Keep removing redundant points from X until X is in reduced form.
2. IF T has height at most ρ,
3. **return** LeafBST(X, T)
4. Let U be the set of vertices lying in the middle layer of T.
5. Compute the split $(X^c, \{X^s_v\}_{v \in U})$ of (X, T) at U.
6. Compute the corresponding sub-trees $(T^c, \{T^s_v\}_{v \in U})$ of T.
7. For each vertex $v \in U$, call to RecursiveBST with input (X^s_v, T^s_v, ρ), and let Y_v be the solution returned by it.
8. Call RecursiveBST with input (X^c, T^c, ρ), and let $\hat{Y}$ be the solution returned by it.
9. Let Z be a point set containing, for each vertex $v \in U$, for each point $p \in X^s_v$, two copies p' and p'' of p with $p'.y = p''.y = p.y$, where p' lies on the left boundary of $S(v)$, and p'' lies on the right boundary of $S(v)$.
10. **return** $Y^* = Z \cup \hat{Y} \cup (\bigcup_{v \in U} Y_v)$

A.4 Analysis

We start by showing that the solution that the algorithm returns is T-special in the following observation, whose proof appears in the full version of the paper.

▶ **Observation 31.** *Assuming that* LeafBST(X,T) *returns a* T*-special solution, the solution* Y^* *returned by Algorithm* RecursiveBST(X,T,ρ) *is a* T*-special solution.*

We next turn to prove that the solution Y^* computed by Algorithm RecursiveBST(X,T,ρ) is feasible. In order to do so, we will use the following immediate observation.

▶ **Observation 32.** *Let* Y^* *be the solution returned by Algorithm* RecursiveBST(X,T,ρ)*, and let* $u \in U$ *be any vertex. Then:*

- *Any point* $y \in Y^*$ *that lies in the interior of* $S(u)$ *must lie on an active row of instance* X^s_u*.*
- *Any point* $y \in Y^*$ *that lies on the boundary of* $S(u)$ *must belong to in* $\hat{Y} \cup Z$*. Moreover, the points of* $\hat{Y} \cup Z$ *may not lie in the interior of* $S(u)$*.*
- *If* R *is an active row for instance* X^s_u*, then set* Z *contains two points, lying on the intersection of* R *with the left and the right boundaries of* $S(u)$*, respectively.*

The following theorem, whose proof is deferred to the full version of the paper, shows that the algorithm returns a feasible solution.

▶ **Theorem 33.** *Assume that the recursive calls to Algorithm* RecursiveBST *return a feasible special solution* $\hat{Y}$ *for instance* X^c*, and for each* $v \in U$*, a feasible special solution* Y_v *for the strip instance* X^s_v*. Then the point set* $Y^* = Z \cup \hat{Y} \cup (\bigcup_{v \in U} Y_v)$ *is a feasible solution for instance* X*.*

In order to analyze the solution cost, consider the final solution Y^* to the input instance X. We distinguish between two types of pints in Y^*: a point $p \in Y^*$ is said to be of type 2 if it was added to the solution by Algorithm LeafBST, and otherwise we say that it is of type 1. We start by bounding the number of points of type 1 in Y^*.

▷ Claim 34. The number of points of type 1 in the solution Y^* to the original instance X is at most $O(\log(\text{height}(T)/\rho)) \cdot \mathsf{OPT}(X)$.

Proof. Observe that the number of recursive levels is bounded by $\lambda = O(\log(\text{height}(T)/\rho))$. This is since, in every recursive level, the heights of all trees decrease by a constant factor, and we terminate the algorithm once the tree heights are bounded by ρ. For each $1 \leq i \leq \lambda$, let $\mathcal{X}_i$ be the collection of all instances in the ith recursive level, where the instances are in the reduced form. Notice that the only points that are added to the solution by Algorithm RecursiveBST directly are the points in the sets Z. The number of such points added at recursive level i is bounded by $\sum_{X' \in \mathcal{X}_i} 2|X'|$. It is now sufficient to show that for all $1 \leq i \leq \lambda$, $\sum_{X' \in \mathcal{X}_i} |X'| \leq O(\mathsf{OPT}(X))$. We do so using the following observation.

▶ **Observation 35.** *For all* $1 \leq i \leq \lambda$, $\sum_{X' \in \mathcal{X}_i} \mathit{OPT}(X') \leq \mathit{OPT}(X)$.

Assume first that the observation is correct. For each instance $X' \in \mathcal{X}_i$, let T' be the partitioning tree associated with X'. From Lemma 28, $|X'| \leq O(\mathsf{OPT}(X'))$. Therefore, the number of type-1 points added to the solution at recursive level i is bounded by $O(\mathsf{OPT}(X))$. We now turn to prove Observation 35.

Proof of Observation 35. The proof is by induction on the recursive level i. It is easy to see that the claim holds for $i = 1$, since, from Observation 25, removing redundant points from X to turn it into reduced form cannot increase $\mathsf{OPT}(X)$.

Assume now that the claim holds for level i, and consider some level-i instance $X' \in \mathcal{X}_i$. Let $(X^c, \{X^s_u\}_{u \in U})$ be the split of (X', T') that we have computed. Then, from Theorem 12, $\sum_{v \in U} \mathsf{OPT}(X^s_v) + \mathsf{OPT}(X^c) \leq \mathsf{OPT}(X')$. Since, from Observation 25, removing redundant points from an instance does not increase its optimal solution cost, the observation follows. ◀

◁

In order to obtain an efficient $O(\log \log n)$-approximation algorithm, we set ρ to be a constant (it can even be set to 1), and we use algorithm LeafBST-2 whenever the algorithm calls to subroutine LeafBST. Observe that the depth of the recursion is now bounded by $O(\log \log n)$, and so the total number of type-1 points in the solution is bounded by $O(\log \log n) \cdot \mathsf{OPT}(X)$. Let $\mathcal{I}$ denote the set of all instances to which Algorithm LeafBST is applied. Using the same arguments as in Claim 34, $\sum_{X' \in \mathcal{I}} |X'| = O(\mathsf{OPT}(X))$. The number of type-2 points that Algorithm LeafBST adds to the solution for each instance $X' \in \mathcal{I}$ is bounded by $O(|X'| \cdot \rho) = O(|X'|)$. Therefore, the total number of type-2 points in the solution is bounded by $O(\mathsf{OPT}(X))$. Overall, we obtain a solution of cost at most $O(\log \log n) \cdot \mathsf{OPT}(X)$, and the running time of the algorithm is polynomial in $|X|$.

Finally, in order to obtain the sub-exponential time algorithm, we set the parameter ρ to be such that the recursion depth is bounded by D. Since the number of active columns in instance X is $c(X)$, and the height of the partitioning tree T is bounded by $2 \log c(X)$, while the depth of the recursion is at most $2 \log(\text{height}(T)/\rho)$, it is easy to verify that $\rho = O\left(\frac{\log c(X)}{2^{D/2}}\right) = \frac{\log c(X)}{2^{\Omega(D)}}$. We use algorithm LeafBST-1 whenever the algorithm calls to subroutine LeafBST. As before, let $\mathcal{I}$ be the set of all instances to which Algorithm LeafBST is applied. Using the same arguments as in Claim 34, $\sum_{X' \in \mathcal{I}} (|X'| + \mathsf{OPT}(X')) = O(\mathsf{OPT}(X))$. For each such instance X', Algorithm LeafBST-1 produces a solution of cost $O(|X'| + \mathsf{OPT}(X'))$. Therefore, the total number of type-2 points in the final solution is bounded by $O(\mathsf{OPT}(X))$. The total number of type-1 points in the solution is therefore bounded by $O(D) \cdot \mathsf{OPT}(X)$ as before. Therefore, the algorithm produces a factor-$O(D)$-approximate solution. Finally, in order to analyze the running time of the algorithm, we first bound the running time of all calls to procedure LeafBST-1. The number of such calls is bounded by $|X|$. Consider now some instance $X' \in \mathcal{I}$, and its corresponding partitioning tree T'. Since the height of T' is bounded by ρ, we get that $c(X') \leq 2^{\rho} \leq 2^{\log c(X)/2^{\Omega(D)}} \leq (c(X))^{1/2^{\Omega(D)}}$. Therefore, the running time of LeafBST-1 on instance X' is bounded by $|X'|^{O(1)} \cdot (c(X'))^{O(c(X'))} \leq |X'|^{O(1)} \cdot \exp\left(O(c(X') \log c(X')\right) \leq |X'|^{O(1)} \cdot \exp\left(c(X)^{1/2^{\Omega(D)}} \cdot \log c(X)\right)$.

The running time of the remainder of the algorithm, excluding the calls to LeafBST-1, is bounded by $\text{poly}(|X|)$. We conclude that the total running time of the algorithm is bounded by $|X|^{O(1)} \cdot \exp\left(c(X)^{1/2^{\Omega(D)}} \cdot \log c(X)\right) \leq \text{poly}(m) \cdot \exp\left(n^{1/2^{\Omega(D)}} \cdot \log n\right)$.