The Geometry of Binary Search Trees*

Erik D. Demaine† Dion Harmon‡ John Iacono§ Daniel Kane¶ Mihai Pătraşcu†

Abstract

We present a novel connection between binary search trees (BSTs) and points in the plane satisfying a simple property. Using this correspondence, we achieve the following results:

1. A surprisingly clean restatement in geometric terms of many results and conjectures relating to BSTs and dynamic optimality.

2. A new lower bound for searching in the BST model, which subsumes the previous two known bounds of Wilber [FOCS'86].

3. The first proposal for dynamic optimality not based on splay trees. A natural greedy but offline algorithm was presented by Lucas [1988], and independently by Munro [2000], and was conjectured to be an (additive) approximation of the best binary search tree. We show that there exists an equal-cost *online* algorithm, transforming the conjecture of Lucas and Munro into the conjecture that the greedy algorithm is dynamically optimal.

1 Introduction

Instance optimality. One of the pillars of theoretical computer science is worst-case analysis: "assume the worst possible data." By this account, binary search trees (BSTs) need $\Theta(\lg n)$ time for a search; in particular, information theory shows that searching[1] for a uniformly random element requires $\Omega(\lg n)$ comparisons on average.

However, no sooner is a proof of a lower bound or conditional hardness discovered than questions of the following flavor pop up: "Do these hard instances actually show up in real life? Can we do better on some interesting, easier instances?" A multitude of measures for instance hardness have been developed, and algorithms attempt to match them for the benefit of easier instances. For BSTs, such measures include the entropy bound [ST85], the working-set bound [ST85], static/dynamic finger bounds [CMSS00, Col00], key-independent optimality [Iac05], the unified bound [Iac01, BCDI07], etc.

The "theoretical dream" that can unify such work is an instance-optimal algorithm [FLN03]: for any instance S, this algorithm would run on S (almost) as fast as any possible algorithm. Unfortunately, instance optimality is often not well-defined. Consider, for example, search algorithms on the pointer machine. If we query elements in an order given by a permutation, the "luckiest" algorithm can just store a linked list of the elements in that particular order, and obtain constant time per query. However, no single algorithm can run in $o(\lg n)$ per query for *all* permutations.

For binary search trees, however, the best running time on a particular sequence is an interesting (and poorly understood) measure for the complexity of the sequence. For example, there exist some very structured, deterministic sequences of n queries (such as the bit-reversal permutation) that provably have a total running time of $\Omega(n \lg n)$ for *any* BST algorithm [Wil89]. In contrast, many other types of sequences can be supported in $o(\lg n)$ time per query.

Most interestingly, however, instance optimality might be possible in the BST model. This question has fascinated researchers ever since STOC'83, when Sleator and Tarjan [ST85] conjectured that their splay tree is such a "best binary search tree."

The BST model. For concreteness, we choose the following model for BSTs, among many choices that are constant-factor equivalent. A search is conducted with a pointer starting at the root, which is free to move about the tree and perform rotations; however, the pointer must at some point in the operation visit the item being searched. The *cost* of a search is simply the total number of distinct nodes in the tree that have been visited by the pointer during the operation.

We measure the total cost of executing a sequence of searches $S = \langle s_1, s_2, \dots s_m \rangle$, where each search s_i is chosen from among the fixed set of n keys in the BST. Let $\mathrm{OPT}(S)$ denote the minimal cost for executing the access sequence S in the BST model, or equivalently, the cost of the best *offline* BST algorithm which knows S a priori. This value is well-defined and its decision version is in NP (by exhibiting a sequence of rotations and pointer moves).

OPEN PROBLEM 1.1. *Can we compute or approximate* $\mathrm{OPT}(S)$ *in polynomial time?*

There are reasons to suspect that computing $\mathrm{OPT}(\cdot)$ exactly may be NP-complete, and obtaining a constant-factor approximation might be doable in near-linear time. For static BSTs (no rotations allowed), the question was addressed by a well-known dynamic-programming algorithm

*Many of the results in this paper appeared in the second author's PhD thesis [Har06].

†MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine,mip}@mit.edu

‡New England Complex Systems Institute, 24 Mount Auburn St., Cambridge, MA 02138, USA, dion.harmon@gmail.com. Work performed while at MIT.

§Department of Computer and Information Science, Polytechnic Institute of New York University, 5 MetroTech Center, Brooklyn, NY 11201, USA, http://john.poly.edu/

¶Department of Mathematics, Harvard University, 1 Oxford St., Cambridge, MA 02139, USA, dankane@math.harvard.edu.

[1] For simplicity, this paper only considers exact and successful searches (not predecessor searches).

of Knuth [Knu71].

Instance optimality (historically called "dynamic optimality" for the special case of BSTs) entails a more stringent requirement: we need to approximately achieve $\mathrm{OPT}(S)$ in an online setting, where the queries are revealed one at a time. We are only interested in the cost of the specified BST operations, not in the planning work the algorithm does outside of the BST model. (Of course, any practical algorithm will need to have very fast computation outside the BST model.)

OPEN PROBLEM 1.2. *Is there an online BST algorithm whose total cost is $O(\mathrm{OPT}(S))$ for all S?*

The best known guarantee is the $O(\lg \lg n)$ competitive ratio achieved recently by Tango trees [DHIP07]. This is also the best proven approximation factor achieved for the offline problem.

Splay trees have been famously conjectured to be dynamically optimal [ST85], but they are not known to have any $o(\lg n)$ competitive ratio. Nonetheless, they are known to have many properties that $\mathrm{OPT}(\cdot)$ has: static optimality [ST85], the working-set bound [ST85], the dynamic-finger bound [CMSS00, Col00], linear traversal [Tar85], near-optimal deque behavior [Pet08, Sun92], and near-optimal splitting [Luc88].

The formulation of Open Problems 1.1 and 1.2 puts the instance-optimality questions in the familiar land of approximation algorithms and competitive analysis. However, the general feel of these problems is quite different from problems about renting skis, because the BST model seems closer to "a model of computation" than just "a constrained problem." From this perspective, instance optimality appears like an ambitious goal of understanding computation in this (admittedly very restrictive) model—a goal that we still seem far from achieving. The appeal of the question lies in the contrast between the scant progress on optimality, and the simplicity of the model. To put this in a different light, we do not find BST optimality interesting *despite* that "the worst case is a just $O(\lg n)$, anyway," but precisely because of it.

1.1 Our Results.

A geometric view. In §2, we present an exact correspondence between the BST model of computation and the following clean question about points in the plane.

Call a set P of points *arborally satisfied* if, for any two points $a, b \in P$ not on a common horizontal or vertical line, there is at least one point from $P \setminus \{a, b\}$ in the axis-aligned rectangle defined by a and b; see Figure 1. We plot an access sequence S in the natural two-dimensional way: $P = \{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$. We show that finding the best BST execution for S is equivalent to finding the minimum cardinality superset $P' \supseteq P$ that is arborally satisfied.

This *geometric view* of BSTs is much easier to work with than BSTs in their natural *arboral* state. While the connection to the geometric view is not technically complicated, we find this conceptual change both compelling and useful; in particular, it has led us to the technical results below. It is quite plausible that this simple transformation will be the most enduring message of our paper.

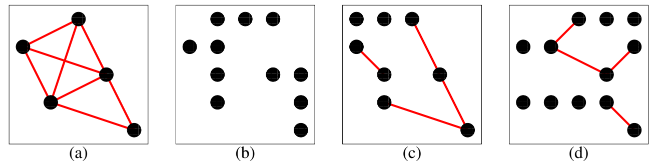


Figure 1: Satisfying point sets. Lines connect pairs of points that define unsatisfied rectangles. (a) is just an access sequence and is not arborally satisfied; (b) is an arborally satisfied superset (the output of GREEDYFUTURE); (c) is a superset of (a) that is not arborally satisfied but is ⧄-satisfied (the output of ⧄-SIGNEDGREEDY); (d) is a superset of (a) that is neither arborally satisfied nor ⧄-satisfied.

Another candidate for dynamic optimality. Lucas [Luc88] presented an offline BST algorithm, GREEDYFUTURE, that intuitively should work very well: when searching for an item, re-arrange the nodes on the search path to minimize the cost of the future searches, by putting the node or subtree containing the next search as close to the root as possible, and recursing to the left and right (see §3 for further details). Intuitively, going off the search path should not help (too much), and the re-arrangement of the path is the best possible. Thus, Lucas conjectured that her algorithm gives a constant-factor approximation for $\mathrm{OPT}(S)$.

More than a decade later, Munro [Mun00] proposed the same algorithm independently. His thought experiments suggested that the only cause for non-optimality is an occasional wrong choice of the root, leading to an unnecessary additive cost of 1 per operation. Thus, Munro conjectured that the cost of GREEDYFUTURE is $\mathrm{OPT}(S) + O(m)$. (This conjecture about additive optimality is made in our precise BST model, because other models differ by constant factors.)

This conjecture has survived computer experiments by Lars Jacobsen and Ian Munro (reported in personal communication), and a few independent experiments by ourselves. The simple example of Figure 2 demonstrates a cost of $\mathrm{OPT}(S) + m/2$.

CONJECTURE 1.1. *On any access sequence S,* GREEDYFUTURE *has cost $O(\mathrm{OPT}(S))$. More strongly, it has cost $\mathrm{OPT}(S) + O(m)$.*

The GREEDYFUTURE algorithm appears like a very sensible and powerful use of future knowledge, with conjectured near-optimality. Thus, GREEDYFUTURE would seem to be the most likely candidate for a separation of online and offline BST algorithms. Emphatically, Lucas and Munro describe their algorithm as something close to the ideal offline optimum that online algorithms like splay trees must try to match.

However, our geometric model allows for a very simple view of GREEDYFUTURE: the algorithm traverses the access sequence with a horizontal sweep line, and at any moment, adds the minimal number of points on the sweep line to make all rectangles below the line arborally satisfied. In this geometric view, any dependence on the "future" is lost, and by (carefully) applying the conversion from the geometric view back to trees, we obtain an *online* algorithm with just a constant factor slow-down (assuming $m \geq n$).

In other words, we show that, if Conjecture 1.1 is true (even in the weak sense), our online version of GREEDY-

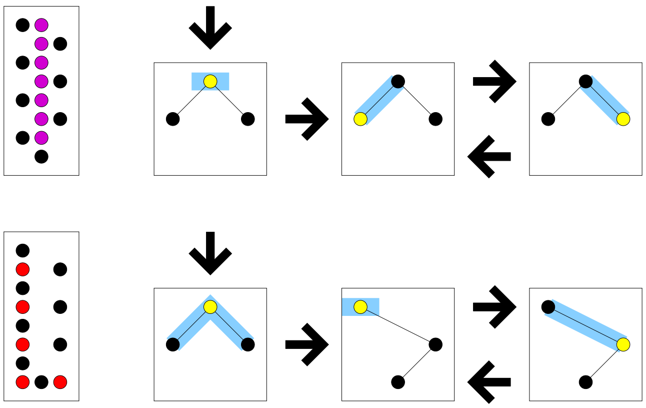


Figure 2: A simple three-node example (which repeats searches for the same key) where the greedy algorithm is suboptimal. The greedy algorithm is on top and the optimal algorithm is on bottom. The geometric view is on the left, and a state diagram of the arboral view is on the right. After the first operation, the optimal algorithm costs 1.5 per search, while the greedy algorithm costs 2. This difference seems to come from the greedy algorithm's unwillingness to pre-emptively visit a node off of the search path, as this prevents it from moving the root, which will never be searched again, out of the way.

FUTURE is dynamically optimal! The only previous proposals for dynamic optimality are the original splay trees [ST85], and the multisplay trees[2] of [WDS06] which are a combination of splay trees and Tango trees [DHIP07]. Unfortunately, the full behavior of splay trees can be described as either "complete magic" or "notoriously hard to understand," depending on the reader's mood. One can thus hope that dynamic optimality will be more approachable, now that there is a proposal not hinging on splay trees.

It should be noted that, despite Conjecture 1.1, our understanding of GREEDYFUTURE is rather embarrassing. We do not even know that its worst case for a sequence of m updates is $O(m \lg n)$! Nor can we prove typical properties that the optimum must have, such as finger bounds, the working set bound, or the entropy bound. Conjecture 1.1 makes a study of these weaker bounds quite interesting, just as the conjectured optimality of splay trees has historically led to intense work on bounds of this flavor.

BST lower bounds. The crucial ingredient for understanding $\mathrm{OPT}(\cdot)$ seems to be constructive lower bounds that algorithms can try to match (exactly or approximately). So far, the only known lower bounds on BSTs have been the two bounds developed by Wilber [Wil89] in FOCS'86. Wilber's first bound was used to show that Tango trees [DHIP07] are $O(\lg \lg n)$-competitive. Wilber's second bound was used to achieve key-independent optimality [Iac05]. Folk wisdom, dating back to a conjecture in Wilber's paper, states that the second bound is stronger, and potentially even $\Theta(\mathrm{OPT}(S))$. However, all attempts to analyze this bound have failed: it is not known that the second bound is better than the first, nor that it gives any nontrivial approximation to $\mathrm{OPT}(\cdot)$.

In §4, we describe a general class of BST lower bounds, called *independent rectangle bounds*, that includes both Wilber bounds as special cases. Pending the appearance of a lower bound outside this class (which would be a very interesting development), one might conjecture that rectangle bounds hold the final answer, i.e., for any S, the best independent rectangle bound for S is $\Omega(\mathrm{OPT}(S))$. This motivates an attempt to find the best bound in the class.

We describe a greedy algorithm, SIGNEDGREEDY, that obtains a constant-factor approximation of the best independent rectangle bound for any sequence S. This bound may be used in attempts to approximate $\mathrm{OPT}(S)$, superseding the two Wilber bounds, as well as side-stepping the issue of comparing the two. By our geometric view, the optimal cost is invariant to flips (running time in reverse) and 90-degree rotations of a permutation access sequence (interchanging the roles of space and time). Compared to the Wilber bounds, our SIGNEDGREEDY bound has the re-assuring feature that it is provably invariant under these geometric transformations (up to constants).

The fascinating feature of the SIGNEDGREEDY algorithm is that it is defined almost identically to the geometric view of GREEDYFUTURE, yet we cannot prove any relation between the upper and lower bound.

NP-hardness of multisearch. In §5, we show that it is NP-complete to find the minimum superset that is arborally satisfied, when the input is a general set of points in the plane. However, this does not show NP-completeness for computing $\mathrm{OPT}(\cdot)$: the set of points corresponding to an access sequence has only one point at each y coordinate, whereas our hard instance has many such points. This general case corresponds to a *multisearch* version of the BST problem: each query is a set S_t, and the algorithm must visit all nodes from S_t (in any order) while executing the query.

A note on independent work. A technical report of Derryberry, Sleator, and Wang [DSW05] also contains a geometric view of the access sequence as points $(1, s_1), (2, s_2), \ldots, (m, s_m)$. Their view only represents the *access sequence* with the goal of arguing about lower bounds, whereas the highlight of our view is to represent the *actions* of any BST algorithm, as an arborally satisfied superset. Representing GREEDYFUTURE in this way was the inspiration for converting it to an online algorithm.

Like us, [DSW05] propose a class of lower bounds that contains both of Wilber's bounds. Ultimately, this class of lower bounds appears to be identical to ours, though the definition is slightly more complicated (they consider rectangles augmented with a "divider" line). The authors do not consider the question of optimizing over this class of lower bounds (the question solved by our SIGNEDGREEDY algorithm).

2 Trees and Arborally Satisfied Points Sets

2.1 Defining the BST Model. As is standard in work on BST optimality, we consider only (successful) searches, not insertion and deletion. The letters n and m always refer to the size of a BST, and the total number of search operations performed on it, respectively. These are fixed global constants, and much of the notation depends on

[2]It is unclear how plausible this proposal is. The authors themselves only state that "as far as we know, multi-splay trees may be dynamically optimal."

these values, either explicitly or implicitly. For simplicity, we denote the ordered values in the BST by the integers $1, 2, \ldots, n$.

We begin with a precise definition of the BST model of computation and its costs:

DEFINITION 2.1. *Given a BST T_1, a subtree τ of T_1 containing the root, and a tree τ' on the same nodes as τ, we say T_1 can be* reconfigured *by an operation $\tau \to \tau'$ to another BST T_2 if T_2 is identical to T_1 except for τ being replaced by τ'. The cost of the reconfiguration is $|\tau| = |\tau'|$.*

DEFINITION 2.2. *Given a search sequence $S = \langle s_1, s_2, \ldots s_m \rangle$, we say a BST algorithm executes S by an execution $E = \langle T_0, \tau_1 \to \tau'_1, \ldots, \tau_m \to \tau'_m \rangle$ if all reconfigurations are valid, and $s_i \in \tau_i$ for all i. For $i = 1, 2, \ldots, m$, define T_i to be T_{i-1} with the reconfiguration $\tau_i \to \tau'_i$. The cost of execution E is $\sum_{i=1}^{m} |\tau_i|$.*

This model is constant-factor equivalent to other BST models. For example, one other way to model a search tree is to view each search operation as starting with a pointer at the root, with the following unit-cost operations supported: move pointer to left child, move pointer to right child, move pointer up, right rotate at pointer, left rotate at pointer. This equivalence holds because any BST can be converted into any other BST with the same nodes in linear time [STT86]. See [Wil89, Luc88] for discussions of alternative models.

2.2 Arborally Satisfied Sets. Switching to the geometric side, a *point* p refers to a point in 2D with integer coordinates $(p.x, p.y)$ such that $1 \le p.x \le n$ and $1 \le p.y \le m$. We use $\Box ab$ to denote the axis-aligned rectangle with corners a and b (viewed as a region including the boundary of the rectangle).

DEFINITION 2.3. *A pair of points (a, b) (or their induced rectangle $\Box ab$) is* arborally satisfied *with respect to a point set P if (1) a and b are orthogonally collinear (horizontally or vertically aligned), or (2) there is at least one point from $P \setminus \{a, b\}$ in $\Box ab$. A point set P is* arborally satisfied *if all pairs of points in P are arborally satisfied with respect to P.*

See Figure 1 for an illustration of satisfaction.

OBSERVATION 2.1. *In an arborally satisfied point set P, for any $a, b \in P$ not orthogonally collinear, there is at least one point from $P \setminus \{a, b\}$ on the sides of $\Box ab$ incident to a, and at least one point on the sides incident to b. (The two points need not be distinct.)*

Proof: Consider any two points $a, b \in P$ that are not orthogonally collinear. Because $\Box ab$ is satisfied, it contains some other point $c \in P$. If c is not on either of the sides of $\Box ab$ incident to a, then we can recurse into $\Box ac$ until we find such a point. Similarly, if c is not on either of the sides of $\Box ab$ incident to b, then we can recurse into $\Box cb$ until we find such a point. $\Box$

We now plot an execution of the BST algorithm in an intuitive way: at time i (row i), we plot all nodes touched in τ_i. The BST model has been chosen to ignore just the right amount of detail (e.g., precise rotations and pointer movements) to make this geometric view easy.

DEFINITION 2.4. *The* geometric view *of a BST execution E is the point set $P(E) = \{(x, y) \mid x \in \tau_y\}$.*

LEMMA 2.1. *The point set $P(E)$ for any BST execution E is arborally satisfied.*

Proof: Assume for contradiction that we can find $a \in \tau_i$ and $b \in \tau_j$, with $i < j$ and $a \neq b$, and yet no other nodes in $[a, b]$ were touched in the closed time interval $[i, j]$. Let c be the lowest common ancestor of a and b in tree T_i. We distinguish two cases:

- $c \neq a$**.** Then c must be touched at time i to get to a ($c \in \tau_i$) and c must have a key value between a exclusive and b inclusive. Contradiction.
- $c = a$**.** Then, at time i, a is an ancestor of b. By assumption that $\Box ab$ is unsatisfied, b is not touched from time i inclusive to j exclusive. Thus a will remain on the access path of b, i.e., a must be an ancestor of b in T_j, and will be touched then ($a \in \tau_j$). Contradiction. $\Box$

2.3 Offline Equivalence. The geometric view seems like a very lossy (non-injective) representation of a BST execution, because it represents only the set of nodes in τ_i, and does not say how these nodes should be reconfigured through rotations. Indeed, the geometric representation does not even mention the sequence τ'. Quite surprisingly, it turns out that the exact shape of the tree is not essential information, and can be reconstructed from just the sets of accessed nodes! In other words, we can reconstruct the execution sequence of any geometric view that satisfies the necessary condition of being arborally satisfied.

LEMMA 2.2. *For any arborally satisfied point set X, there exists a BST execution E with $P(E) = X$. We call E the* arboral view *of X, and write $P^{-1}(X) = E$.*

Proof: We describe an algorithm for the reverse transformation $P^{-1}(\cdot)$, as illustrated in Figure 3. Define the *next access time* $N(x, i)$ of x at time i to be the minimum y coordinate of any point in X on the ray from (x, y) to (x, ∞). If there is no such point, $N(x, i) = \infty$.

Let T_i be the treap defined on all points $(x, N(x, i))$. Recall that a treap is a BST on the first coordinate and a heap on the second, where ties are broken arbitrarily. Thus, T_i is a valid BST on the n data values, chosen to satisfy the heap property according to the next access time.

Let τ_i be the points in X with $y = i$. By the treap property of T_i, these must form a connected subtree of T_i that includes the root (because i is the minimum possible access time $N(\star, i)$). Now, form T_{i+1} by re-arranging the nodes in τ_i to form a treap based on the next access time $(x, N(x, i+1))$. All we need to show is that T_{i+1} is a treap on $(x, N(x, i+1))$. The BST property trivially holds by construction, so we look at the heap property. It suffices to show that the heap property holds between every parent/child pair (q, r) in T_i. If both were in τ_i, the heap property follows by construction, and if both were outside τ_i, the heap property holds because neither their next access times nor their parent/child relationship changed from i to $i+1$. We are left with the case $q \in \tau_i$ and $r \notin \tau_i$. But if the heap property

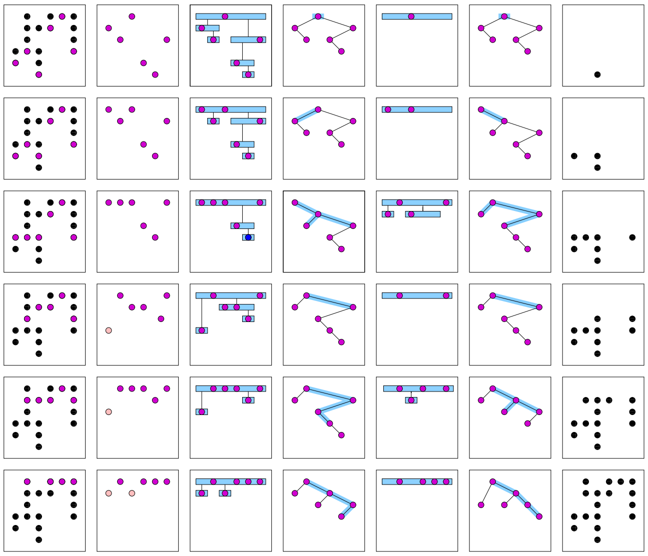


Figure 3: Illustration of the transformations between the tree and geometric view. Columns from left to right: (1) The point set X; on row i (time i for the sweep line algorithm), the light nodes are the appearances that define $N(x, i)$. (2) The light nodes from column 1 have been redrawn and flipped vertically for easy conversion to a tree; the even lighter nodes have $N(x, i) = \infty$ and appear at the bottom. (3) A general treap, constructed based on these next values. (4) The tree T_{i-1}. (5) The treap formed by the next access time of the nodes of τ_i at time $i+1$; τ' is an arbitrary binaryification of this general treap. (6) The tree T_i.

The shaded nodes and edges in columns 4 and 6 indicate τ_i and τ_i'. Note that these nodes are exactly the same nodes as the root of the treap in column 3, and the row being swept in column 1. The tree T_i (column 6) is obtained by replacing the nodes of τ_i (shaded nodes) in T_{i-1} with arbitrary binarification of the treap in column 5. In column 7, we draw the shaded τ nodes from columns 4 or 6, and obtain the original point set from column 1.

is violated in T_{i+1}, the rectangle from (q, i) to $(r, N(r, i))$ violates Observation 2.1: the vertical side at $x = q$ is empty, because $N(q, i + 1) > N(r, i)$ by the assumption that the heap property is violated. The horizontal side at $y = i$ is also empty, because otherwise r could not be the child of q (i.e., a point on that side would become the lowest common ancestor of q and r in T_{i+1}). Contradiction. □

Let the *geometric view* of an access sequence S be the set of points $P(S) = \{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$. Lemmas 2.1 and 2.2 have shown that the arboral statement "E executes S" is equivalent to the geometric statement "$P(S) \subseteq P(E)$." Letting $\mathrm{minASS}(S)$ be the size of the smallest arborally satisfied superset of $P(S)$, we have $\mathrm{OPT}(S) = \mathrm{minASS}(P(S))$. Thus, Open Problem 1.1 is equivalent to designing algorithms for finding the minimum arborally satisfied superset.

2.4 Online Equivalence. Above, we gave a combinatorial equivalence of tree executions and arborally satisfied sets, effectively characterizing offline BST algorithms. We now wish to strengthen this characterization to *online* BST algorithms, which must produce the transformation $\tau_i \to \tau_i'$ after each s_i is revealed, with no knowledge of future accesses.

DEFINITION 2.5. *The* online arborally satisfied superset (online ASS) *problem is to design an algorithm that receives a set of points $\{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$ incrementally. After receiving point i, the algorithm must output a set P_i of points on the line $y = i$ such that $\{(s_1, 1), (s_2, 2), \ldots, (s_i, i)\} \cup P_1 \cup P_2 \cup \cdots \cup P_i$ is arborally satisfied. The cost of the algorithm is $\sum_{i=1}^{n} |P_i|$.*

An online BST algorithm gives an online ASS algorithm by the standard geometric representation (Lemma 2.1). The reverse statement is not obvious, because Lemma 2.2 needs powerful knowledge of the future: it reconstructs the shape of the tree by imposing a heap order based on future access times. However, using the following interesting concept, we will be able to "guess" the shape of the tree dynamically, with only a constant-factor loss in the running time:

DEFINITION 2.6. *A* split tree *is an abstract data type implementing two operations in the BST model:*

MAKETREE$(x_1, x_2, \ldots, x_n)$ *creates an n-node BST on the given values.*

SPLIT(x) *moves x to the root of the tree and then deletes it, leaving the left and right subtrees as valid split trees.*

We show below (§2.5) that split trees can be implemented with worst-case cost $O(n)$ for MAKETREE and an arbitrary sequence of n SPLITs. Using this fact, we can show a constant-factor equivalence of online ASS and online BST algorithms.

LEMMA 2.3. *For any online ASS algorithm A, there exists an online BST algorithm A' such that, on any access sequence, the cost of A' is bounded by a constant times the cost of A.*

Proof: The main idea is that, whenever we would like to put a set of elements in some unknown future order, we avoid making any decisions and keep them in a split tree. The construction that we are led to can be seen as inverting the proof of Lemma 2.2, because split trees are stored in heap order by the previous access (instead of the next access).

Formally, let $\rho(x, i)$ be the last access of x before time i, that is, the y coordinate of the highest point on the ray from (x, y) to $(x, -\infty)$. If there is no such point, $\rho_P(x, i) = -\infty$. Let G_i be a *general* treap defined on all points $(x, N(x, i))$, i.e., a BST on the x coordinates and a general heap on $N(x, i)$. In a general heap, any ties result in a supernode with more than one key value. The supernode is implemented in the BST model by storing the keys in a split tree.

Now consider how this structure can change from time i to $i + 1$. All points on row $y = i + 1$ will have $\rho(x, i + 1) = i + 1$, and they will be moved from wherever they are in G_i to the root of G_{i+1}.

The key property that follows from arboral satisfaction is that, whenever a key x from node v is accessed, its predecessor and successor in its parent must also be accessed. This means that we can move all the appropriate values to the root, in a single root-to-leaf traversal of the treap. At each supernode, we SPLIT the nodes being searched to form a new singleton node and two children supernodes. Now, all the singleton nodes that we have collected on the path can be turned into the root supernode by calling MAKETREE. Because each SPLIT takes constant time amortized, and MAKETREE takes linear time, the cost of simulating the accesses at $y = i$ is a constant times the number of points on that line. □

2.5 Split Trees. The split-tree data type has been studied before by Lucas [Luc88], who was interested in the performance of splay trees under these operations (where MAKETREE initializes the splay tree to a path, for example). She conjectured that the running time for splay trees over any sequence of n SPLITs is $O(n)$, and managed to prove that it is $O(n \cdot \alpha(n))$. Here we construct another tree with provably linear performance:

LEMMA 2.4. *There is a split-tree data structure supporting* MAKETREE *and any sequence of* n SPLIT*s in total time* $O(n)$.

Proof: We start the design from a 2-3-4 tree. Assume for the moment that we have a pointer to the minimum element in such a tree and we wish to unlink the smallest k items from the tree. The unlinking can be done in $O(\lg k)$ time, including the rebalancing of the $O(\lg k)$ affected nodes. The rebalancing proceeds in the typical way: first attempt to borrow from a neighbor at $O(1)$ cost, and if that fails, merge with the neighbor. This merging could cascade up the tree, but in total we can perform only $O(n)$ node merges.

Continuing our wishful thinking, assume that each SPLIT can choose to start either at the minimum or at the maximum. Then splitting an n-node tree into trees of size $n_1 + n_2 = n$ will take time $O(\lg \min\{n_1, n_2\})$. It is a standard exercise to show that the sum of these costs over any sequence of splits in $O(n)$, by using the potential function $\phi(n) = n - \lg n$.

We must now simulate this ideal algorithm in the BST model (making it a proper binary tree, and only starting at the root). Conceptually, the simulation is very easy: the BST model stores the comparison tree of what we wish to do on the 2-3-4 tree. Adding some simple markers to every node allows us to maintain the comparison tree when the 2-3-4 tree is changed locally. Then, SPLIT only needs to rotate the split element to the root, obtaining two subtrees in the correct representation. □

3 Greedy Future

We begin by describing the offline BST algorithm GREEDYFUTURE, proposed by Lucas [Luc88] and Munro [Mun00]; refer to Figure 4.

ALGORITHM 3.1. *The* GREEDYFUTURE *algorithm follows two principles: (1) only touch the nodes on the search path,* and *(2) to re-arrange the search path, move the next item to be accessed as high as possible, and recurse.*

More formally, let τ_i be the search path for s_i in T_i. If the next search in the access sequence is in τ_i, make that node the root and recurse on both sides. If the next search is in a subtree hanging from the path τ_i, re-arrange the predecessor and successor from τ_i as the root and the right child of the root; then recurse on the parts of the tree that have yet to be specified.

The main "greedy" aspect of the algorithm is that it makes the locally cheapest decision of only touching the search path. The re-arrangement of the search path appears to be an optimal (and powerful) use of knowledge about the future. Thus, the main intuition behind the conjectured optimality of this algorithm is this: "there is no need to access anything off the search path, as you could just access this stuff later." As Figure 2 shows, this is not entirely true, because not going outside the path may force you to keep some useless nodes on the path. The worst-case example that has been found [Mun00] is to search the leaves of a complete tree in bit-reversal order (repeated many times). The GREEDYFUTURE algorithm will not change the tree, despite all internal nodes never being searched, giving a cost of $\lg n$. The optimal cost is $\lg \frac{n}{2} + o(1)$ as the number of repetitions goes to infinity, because the useless nodes can be rotated below the useful ones for a fixed additional cost. Because a worse example was never found, one may conjecture that GREEDYFUTURE's cost is at most $\mathrm{OPT}(X) + m$.

Looking at GREEDYFUTURE in our geometric view hides the ugly details of re-arranging the path, and transforms the algorithm into the following natural greedy algorithm for the ASS problem:

ALGORITHM 3.2. *Sweep the point set with a horizontal line by increasing y coordinate. At time i,* GREEDYASS *places the minimal number of points at $y = i$ to make the point set up to $y \le i$ arborally satisfied. This minimal set of points is uniquely defined: for any unsatisfied rectangle formed with (s_i, i) in one corner, add the other corner at $y = i$.*

The proof of the transformation is rather straightforward; refer to Figures 1 and 4. Because of the way GREEDYFUTURE re-arranges the search path, a future query for an element off the path touches only two nodes on the current search path (or only one node, if the query is left of the minimum or right of the maximum on the path). These two nodes form unsatisfied rectangles, left and right of the query, so they are also "touched" by GREEDYASS.

The key observation is that GREEDYASS is an *online* ASS algorithm, because its decisions depend only on the past (points at lower y coordinates). This means that the only important part of the algorithm is the greedy decision to only touch the search path (equivalently, to add the minimum number of points on the sweep line). Then, by Lemma 2.3, the online GREEDYASS can be turned back into an online BST algorithm with equal cost (up to a constant factor) as the original GREEDYFUTURE!

This onlinification of a "maximally offline" algorithm seems to suggest that offline algorithms cannot asymptotically beat the best online algorithms in the BST model, i.e., that dynamic optimality is possible.

4 Lower Bounds

This section discusses lower bounds for BST algorithms in our geometric view. Equivalently, these are lower bounds on the size of the minimum satisfied superset, which an approximation algorithm can try to match. Our main results are (1) a class of lower bounds that includes all known lower bounds ideas, in particular Wilber's bounds; and (2) a greedy

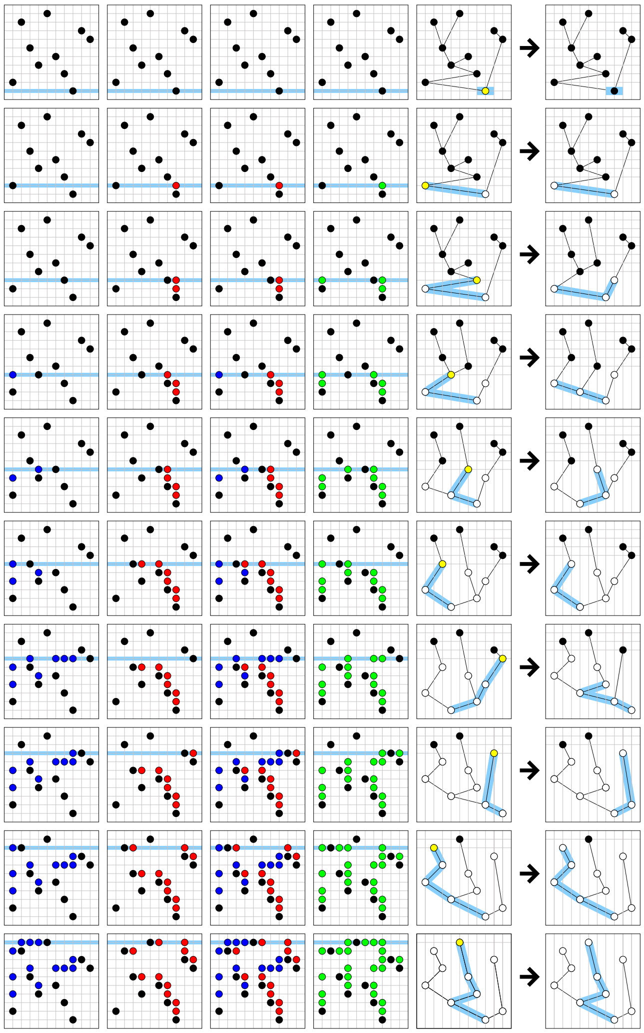


Figure 4: Black nodes in the first four columns illustrate a search sequence X, while the lighter nodes illustrate the incremental construction of the greedy algorithms: $\mathrm{add}_{\boxslash}(X)$ in column 1, $\mathrm{add}_{\boxbslash}(X)$ in column 2, $\mathrm{add}_{\boxslash}(X) \cup \mathrm{add}_{\boxbslash}(X)$ in column 3, and GREEDYFUTURE in column 4. The first three columns are lower bounds, while the fourth column is an upper bound. The fifth and sixth columns illustrate GREEDYFUTURE in the arboral view. At each step, the yellow node denotes the query. The search path (blue) is reconfigured to bring the next node among the unsearched nodes (black) as close to the root as possible (and recursively, bring the second-to-next node as closest to the root as possible, etc.). Given multiple choices that obey this rule, an arbitrary choice is made. Observe that exactly the same nodes are shaded in blue in each $G(X)$ frame (4th column) and future greedy frame (5th and 6th). These two algorithms are identical under the tree-geometric transformations.

algorithm, SIGNEDGREEDY, that gives the best lower bound in this class, up to a factor of two. SIGNEDGREEDY is strikingly similar to GREEDYASS, though we cannot prove a formal connection between these lower and upper bounds.

To simplify proofs, this section assumes that the access sequence S is a permutation, i.e., $m = n$ and each item

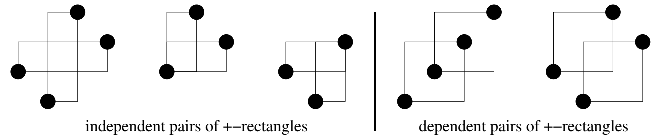


Figure 5: All combinatorially distinct cases when overlapping $\boxslash$-rectangles are independent / dependent.

is searched exactly once. This restriction captures all of the complexity of the problem, and can be removed at the expense of some additional casework. Geometrically, this restriction says that the point set $X = P(S)$ is in "general position" from an orthogonal perspective (no two points are orthogonally collinear).

4.1 Independent Rectangle Bounds. We call two rectangles $\square ab$ and $\square cd$ *independent* (in X) if the rectangles are not arborally satisfied and no corner of either rectangle is strictly inside the other rectangle; see Figure 5. It turns out that independent sets of rectangles are lower bounds:

CLAIM 4.1. *If a point set X contains an independent set I of rectangles, then $\mathrm{minASS}(X) \geq |I|/2 + |X|$. In particular, if $X = P(S)$ for an access sequence S, then $\mathrm{OPT}(S) \geq |I|/2 + |S|$.*

The proof of this claim is deferred (see Theorem 4.1 below) until we can develop some required machinery. We now demonstrate that independent rectangle bounds are a rich class by showing that they contain Wilber's two bounds [Wil89] as special cases. We construct independent rectangle sets equivalent to each Wilber bound, but do not formally prove the equivalence as the precise definitions of Wilber's bounds are too cumbersome to repeat here. In fact, one motivation of our geometric lower bounds is to give a clear independent presentation of those bounds.

The following construction corresponds to Wilber's first bound:

ALGORITHM 4.1. *Let $L = \langle \ell_1, \ell_2, \ldots \rangle$ be any sequence of vertical lines. To define an independent set of unsatisfied rectangles, we first consider ℓ_1, and examine the points of X sorted by their y coordinate. For every pair of consecutive points that switch sides of ℓ_1 (i.e., p_i is left and p_{i+1} is right, or vice versa), we output the rectangle between the two. When we are done with ℓ_1, we recurse among all points and subsequence of lines to the left of ℓ_1, and then recurse among all points and subsequence of lines to the right of ℓ_1.*

The fact that the output rectangles are independent is immediate by construction. The canonical choice for the set of vertical lines are the lines at x coordinates $\langle \frac{1}{2}n, \frac{1}{4}n, \frac{3}{4}n, \frac{1}{8}n, \frac{3}{8}n, \frac{5}{8}n, \frac{7}{8}n, \frac{1}{16}n, \ldots \rangle$.

The next construction corresponds to Wilber's second bound.

ALGORITHM 4.2. *Consider each point p of X in increasing order by y coordinate. Consider the orthogonal envelope of all points in the quadrant $\{(x,y) \mid x < p.x, y < p.y\}$, and the orthogonal envelope of all points in the quadrant*

$\{(x, y) \mid x > p.x, y < p.y\}$. *(The* orthogonal envelope *is the set of points that are not orthogonally dominated by any other point, when looking from p.) Merge the two envelopes and sort them by y coordinate. Now draw a rectangle between any pair of consecutive points switching between the left and right quadrants.*

Again, the independence of the rectangles follows easily by construction.

The exact relationship between the two Wilber bounds remains unknown, though Wilber conjectured that the second one is better. It is also unclear how the bounds are affected by 90-degree rotations of the point set representing the access sequence and, for the second bound, by flips. Computer search reveals many examples where the bounds change slightly, and proving that they change by only a constant factor seems daunting. This is not good news for the potential optimality of the Wilber bounds, because our geometric view makes it clear that $\mathrm{minASS}(\cdot)$ is invariant under these geometric transformations.

Instead of concentrating on the Wilber bounds, it seems sensible to focus on the maximum independent rectangle bound that can be formed on the point set X, which we denote $\mathrm{maxIRB}(X)$. This lower bound is certainly at least as strong as either of the Wilber bounds, and must be invariant to flips and rotations, because the class of independent rectangles bounds is closed under these transformations.

While we cannot compute $\mathrm{maxIRB}(X)$ efficiently, we will be able to give a simple greedy algorithm, SIGNEDGREEDY, that approximates it to within a constant factor.

4.2 Signing Satisfaction. Figure 5 suggests a somewhat unmixable nature of rectangles defined by points at their southeast and northwest corners, and rectangles defined by points at their southwest and northeast corners. Motivated by this idea, we introduce signs to the common concepts that we have developed so far. We say that a rectangle $\Box ab$ defined by two points a, b is a $\boxslash$*-rectangle* ($\boxbslash$*-rectangle*) if the slope of the line $\overline{ab}$ is positive (negative). Below, all statements and definitions using $\boxslash$-rectangles have symmetric statements for $\boxbslash$-rectangles, which we do not explicitly state.

A point set is $\boxslash$*-satisfied* if every pair of points (a, b) that form a $\boxslash$-rectangle $\Box ab$ is arborally satisfied; see Figure 1. In other words, $\boxbslash$-rectangles need not be satisfied for $\boxslash$-satisfaction. Let $\mathrm{minASS}_{\boxslash}(X)$ be the size of the minimum $\boxslash$-satisfied superset of X. We propose the following greedy strategy for computing $\mathrm{minASS}_{\boxslash}(X)$, which is nothing more than GREEDYASS that ignores $\boxbslash$-rectangles:

ALGORITHM 4.3. *Sweep the point set X with a horizontal line by increasing y coordinate. When considering point p on the sweep line, for each unsatisfied $\boxslash$-rectangle formed by x and a point below the sweep line, add the rectangle's northwest corner on the sweep line to make it satisfied. Let $\mathrm{add}_{\boxslash}(X)$ be the final set of added points (excluding X).*

Refer to Figure 4 for a sample execution. Our main interest in this algorithm is not related to computing $\mathrm{minASS}_{\boxslash}(\cdot)$, but to the following connection to independent rectangle bounds:

LEMMA 4.1. *For any X, there exists an independent set of $\boxslash$-rectangles $\mathrm{IRB}_{\boxslash}(X)$ with $|\mathrm{IRB}_{\boxslash}(X)| = |\mathrm{add}_{\boxslash}(X)|$.*

Proof: For every point q in $\mathrm{add}_{\boxslash}(X)$, let $R(q)$ be the $\boxslash$-rectangle that q satisfied, i.e., $R(q) = \Box rs$ where r is the point below q in X and s is the point to the right of q in X. By construction of $\mathrm{add}_{\boxslash}(X)$, $R(q) = R(t)$ if and only if $q = t$. Let $\mathrm{IRB}_{\boxslash}(X) = \{R(q) \mid q \in \mathrm{add}_{\boxslash}(X)\}$, which has $|\mathrm{add}_{\boxslash}(X)| = |\mathrm{IRB}_{\boxslash}(X)|$. To show that the rectangles in $\mathrm{IRB}_{\boxslash}(X)$ are independent, we note that by construction of $\mathrm{add}_{\boxslash}(X)$, no point in $\mathrm{add}_{\boxslash}(X) \cup X$ is strictly inside any rectangle in $\mathrm{IRB}_{\boxslash}(X)$, and that the northwest corners of all rectangles in $\mathrm{IRB}_{\boxslash}(X)$ are in the set $\mathrm{add}_{\boxslash}(X) \cup X$. $\Box$

Combining with Lemma 4.5 below, $|\mathrm{add}_{\boxslash}(X)| + |X|$ and $|\mathrm{add}_{\boxbslash}(X)| + |X|$ are thus lower bounds on $\mathrm{minASS}(X)$ and hence the best BST algorithm $\mathrm{OPT}(S)$ where $X = P(S)$. We define SIGNEDGREEDY by the obvious strategy: run both versions of Algorithm 4.3 and output $\max\{|\mathrm{add}_{\boxslash}(X)|, |\mathrm{add}_{\boxbslash}(X)|\} + |X|$. This shows an interesting duality, by which the signed version of the greedy algorithm gives lower bounds, in contrast to the unsigned GREEDYFUTURE giving upper bounds.

Another interesting contrast is that we can prove optimality of $\mathrm{add}_{\boxslash}(X)$, up to constant factors, unlike the unsigned GREEDYFUTURE:

LEMMA 4.2. *For any X,* $\mathrm{minASS}_{\boxslash}(X) = |\mathrm{add}_{\boxslash}(X)| + |X|$.

Proof: Follows from Lemma 4.5 below and the fact that, by construction, all rectangles in $\mathrm{add}_{\boxslash}(X)$ are $\boxslash$-rectangles. $\Box$

4.3 How Good is SIGNEDGREEDY? We are now going to show that the lower bound output by SIGNEDGREEDY is at least $\frac{1}{4}\mathrm{maxIRB}(X) + \frac{1}{2}|X|$, making it within a constant factor of the best independent rectangle bound. The proof, while not technically complicated, is rather subtle. It begins with the following unintuitive definition:

DEFINITION 4.1. *A superset Z of X is* $\boxtimes$-satisfied *with respect to X if there exist subsets $Z_{\boxslash}$ and $Z_{\boxbslash}$ of Z such that $Z_{\boxslash} \cup X$ is $\boxslash$-satisfied and $Z_{\boxbslash} \cup X$ is $\boxbslash$-satisfied. Let $\mathrm{minASS}_{\boxtimes}(X)$ be the size of the smallest $\boxtimes$-satisfied set Z with respect to X.*

Note that $Z_{\boxslash}$ and $Z_{\boxbslash}$ need not be disjoint; in fact, to minimize $|Z|$ the two parts might overlap significantly. On the other hand, any $\boxslash$-satisfied superset can be combined with any $\boxbslash$-satisfied superset to give a $\boxtimes$-satisfied superset, so

$$(4.1)\quad \mathrm{minASS}_{\boxtimes}(X) \le \mathrm{minASS}_{\boxslash}(X) + \mathrm{minASS}_{\boxbslash}(X).$$

Furthermore, any arborally satisfied set Y is also $\boxtimes$-satisfied (with $Z_{\boxslash} = Z_{\boxbslash} = Y$), so

$$(4.2)\qquad \mathrm{minASS}_{\boxtimes}(X) \le \mathrm{minASS}(X).$$

Therefore, to prove Claim 4.1, it suffices to show the following (see Section 4.4 below for the proof):

THEOREM 4.1. *If X contains an independent set I of rectangles, then* $\mathrm{minASS}_{\boxtimes}(X) \geq |I|/2 + |X|$.

Next, to prove approximate optimality of SIGNEDGREEDY, we make the following chain of deductions:

$$\begin{array}{ll}
\frac{1}{2}\max\{\text{Wilber I}(X), \text{Wilber II}(X)\} + |X| & \} \text{ Algorithms 4.1, 4.2} \\
\leq \frac{1}{2}\,\mathrm{maxIRB}(X) + |X| & \} \text{ Theorem 4.1} \\
\leq \mathrm{minASS}_{\boxtimes}(X) & \} \text{ Equation 4.1} \\
\leq \mathrm{minASS}_{\boxslash}(X) + \mathrm{minASS}_{\boxbslash}(X) & \} \text{ def. of } \mathrm{minASS}_{\boxslash} \\
\leq |\mathrm{add}_{\boxslash}(X)| \;+\; |\mathrm{add}_{\boxbslash}(X)| + 2\,|X| & \} \text{ Lemma 4.1} \\
= |\mathrm{IRB}_{\boxslash}(X)| \;+\; |\mathrm{IRB}_{\boxbslash}(X)| + 2\,|X| & \} \text{ def. of maxIRB} \\
\leq 2\,\mathrm{maxIRB}(X) + 2\,|X| & \} \; 2 \leq 4 \\
\leq 2\,\mathrm{maxIRB}(X) + 4\,|X| & \} \text{ Theorem 4.1, again} \\
\leq 4\,\mathrm{minASS}_{\boxtimes}(X) & \} \text{ Equation 4.2} \\
\leq 4\,\mathrm{minASS}(X). &
\end{array}$$

What exactly is going on here? The unintuitive quantity $\mathrm{minASS}_{\boxtimes}(X)$ is sandwiched due to its dual nature as a lower bound on $\mathrm{minASS}(X)$ and as an upper bound on the best independent rectangle bound $\mathrm{maxIRB}(X) + |X|$. The Tango BST [DHIP07] establishes that $\mathrm{minASS}(X) \leq \text{Wilber I}(X) \cdot O(\lg\lg n)$, so the inequality chain cannot be too loose. In the middle, we obtain a bound on the output of SIGNEDGREEDY: $\max\{|\mathrm{add}_{\boxslash}(X)|, |\mathrm{add}_{\boxbslash}(X)|\} + |X| \geq \frac{1}{2}(|\mathrm{add}_{\boxslash}(X)| + |\mathrm{add}_{\boxbslash}(X)|) + |X| \geq \frac{1}{4}\mathrm{maxIRB}(X) + \frac{1}{2}\,|X|$.

4.4 Proof of Theorem 4.1.

LEMMA 4.3. *Suppose we are given a $\boxslash$-satisfied point set Y with integer x coordinates, a $\boxslash$-rectangle $\square ab$ with $a, b \in Y$, and a vertical line ℓ at a non-integer x coordinate strictly between $a.x$ and $b.x$. Then we can find two points $p, q \in Y$ in the rectangle $\square ab$ such that $p.y = q.y$, p is left of ℓ, q is right of ℓ, and there are no points in Y on the horizontal segment connecting p to q.*

Proof: Let $p \in Y$ be the topmost rightmost point in $\square ab$ to the left of ℓ. (Such a point exists because a is a candidate.) Let $q \in Y$ be the bottommost leftmost point in $\square ab$ to the right of ℓ and at or above p. (Such a point exists because b is a candidate.) By construction, p is left of ℓ, q is right of ℓ, and there are no points in $\square ab$ strictly between p and q in x coordinate. It remains to show that $p.y = q.y$. If $p.y \neq q.y$, then by construction $p.y < q.y$, making $\square pq$ an unsatisfied $\boxslash$-rectangle, contradicting that Y is $\boxslash$-satisfied. $\square$

By symmetry, Lemma 4.3 also holds for $\boxbslash$-rectangles in $\boxbslash$-satisfied sets.

LEMMA 4.4. *Given an independent set I of rectangles in a set X such that each point has a distinct integer x coordinate, there exists a rectangle $\square ab \in I$ and a vertical line ℓ at a non-integer x coordinate strictly between $a.x$ and $b.x$ such that, inside $\square ab$, ℓ does not intersect the interior of any rectangle from $I \setminus \{\square ab\}$.*

Proof: We claim that $\square ab$ can be chosen to be any rectangle in I that does not intersect any wider rectangles in I, for example, the widest rectangle in I. Assume by symmetry that $\square ab$ is a $\boxslash$-rectangle, and that $a.x < b.x$. Any rectangle that intersects $\square ab$ in their interiors cannot have a corner interior to $\square ab$, by independence, so such a rectangle must either intersect both the left and right edges of $\square ab$ or intersect both the top and bottom edges of $\square ab$ (or both). But, because $\square ab$ is (locally) maximally wide, only the latter case is possible. Thus all rectangles interior-overlapping $\square ab$ do so in its entire y extent from bottom edge to top edge.

Now we decompose rectangles interior-overlapping $\square ab$ into three types: (1) those that have a as a corner, (2) those that have b as a corner, and (3) those that have neither a nor b as a corner. All type-1 rectangles must be strictly left of all type-2 rectangles (in horizontal projection): if two were to overlap in their interiors, the corner of the type-1 rectangle on the bottom edge of $\square ab$ other than a itself would be interior to the type-2 rectangle, violating independence, and by distinctness of x coordinates, they cannot overlap on their boundaries either. If a type-3 rectangle intersects a type-1 or type-2 rectangle, then the former rectangle must be narrower than the latter and cross its top and bottom sides: otherwise, by independence, the type-3 rectangle would pierce the left and right sides and then have to be wider than $\square ab$, a contradiction.

Thus we can decompose the horizontal span of $\square ab$ into a range starting at a containing type-1 rectangles and any overlapping type-3 rectangles, then a range of type-3 rectangles that intersect neither type-1 nor type-2 rectangles, followed by a range ending at b containing type-2 rectangles. (Any of these ranges may actually contain no rectangles.) By distinctness of x coordinates, there must be a positive-size gap between any adjacent pair of these ranges, and we can choose the line ℓ to be in one of these gaps. For example, we can choose ℓ to have x coordinate $\frac{1}{2}$ beyond the rightmost edge of any type-1 rectangle (or $a.x$ if no such rectangle exists). $\square$

LEMMA 4.5. *Given an independent set I of $\boxslash$-rectangles in a point set X, any $\boxslash$-satisfied superset Y of X must have cardinality at least $|I| + |X|$.*

Proof: We apply Lemma 4.4 to find a rectangle $\square ab$ in I and a vertical line ℓ piercing $\square ab$ with the property that no other rectangle in I intersects ℓ interior to $\square ab$. Then we apply Lemma 4.3 to find two points p, q horizontally adjacent in Y and on opposite sides of ℓ in $\square ab$. We mark this pair (p, q) with rectangle $\square ab$. Then we remove $\square ab$ from I and repeat the process, until we have marked a pair of horizontally adjacent points in Y for every rectangle in I.

Whenever we remove a rectangle $\square ab$ from I, if p and q are not on the top or bottom sides of $\square ab$, then p and q do not simultaneously belong to any other rectangle in I, so they will never be marked again. On the other hand, if p and q are on the top (bottom) side of $\square ab$, then p and q are neither interior nor on the top (bottom) side of any other rectangle in I. Furthermore, because all rectangles in I are $\boxslash$-rectangles and coordinates in X are distinct, the top side of no rectangle in I coincides even partially with the bottom side of a rectangle in I. Thus, each pair of horizontally adjacent points in Y can be marked at most once.

Finally, by distinctness of y coordinates in X, at most one point in a pair of horizontally adjacent points in Y can belong to X. Therefore the number of points in $Y \setminus X$ is at least the number of rectangles in I, proving the lemma. □

Again, by symmetry, Lemma 4.5 holds for ⧅-rectangles in ⧅-satisfied sets as well.

Proof of Theorem 4.1: Let Z be any ⊠-satisfied set with respect to X, where $Z_{⧄} \cup X$ is ⧄-satisfied and $Z_{⧅} \cup X$ is ⧅-satisfied. Let $I_{⧄}$ denote the ⧄-rectangles in the independent set I, and similarly let $I_{⧅}$ denote the ⧅-rectangles in I. By two applications of Lemma 4.5, $|Z_{⧄} \cup X| \geq |I_{⧄}| + |X|$ and $|Z_{⧅} \cup X| \geq |I_{⧅}| + |X|$. Suppose by symmetry that $|I_{⧄}| \geq |I_{⧅}|$, so $|I_{⧄}| \geq |I|/2$. Therefore, $|Z| \geq |Z_{⧄} \cup X| \geq |I_{⧄}| + |X| \geq |I|/2 + |X|$ as desired. □

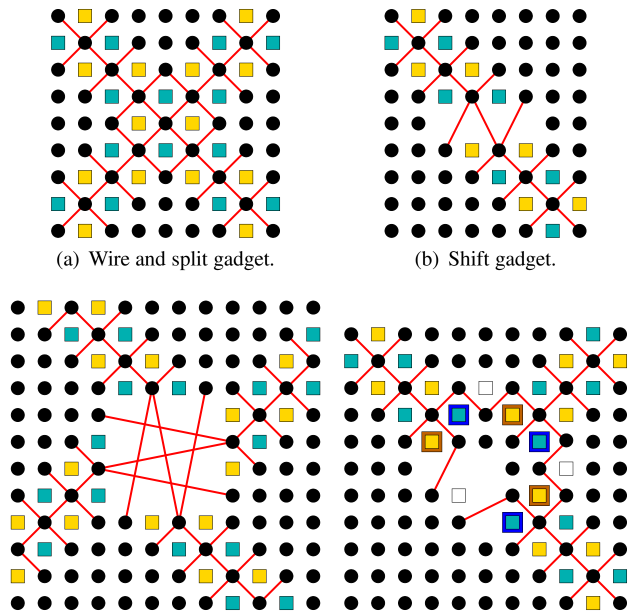
5 Multisearch Is NP-Complete

We do not know whether $\mathrm{minASS}(X)$ can be computed exactly in polynomial time when $X = P(S)$ for an access sequence S, so that no two points share a y coordinate. We can show, however, that the general problem of computing $\mathrm{minASS}(X)$, where the set X may have multiple points at every y coordinate, is NP-complete. This is equivalent to the multisearch problem described in the introduction: given a sequence of sets $S_1, S_2, \ldots, S_m$ of elements, visit a subtree that includes all elements from S_1, then visit such a subtree for S_2, etc. The equivalence of Lemmas 2.1 and 2.2 carry over unchanged for multisearch.

THEOREM 5.1. *It is NP-complete to compute the minimum arborally satisfied superset of a set of points in the plane.*

Proof: The decision version is contained in NP by exhibiting a superset. Figure 6 illustrates the NP-hardness reduction from Not-All-Equal 3SAT. □

Unfortunately, this reduction seems quite far from being relevant to constant-factor approximability, or the restricted problem with one point per y coordinate.



(a) Wire and split gadget. (b) Shift gadget.

(c) Crossover gadget. (d) Not-All-Equal clause gadget.

Figure 6: NP-hardness reduction from Not-All-Equal 3SAT to the offline multiaccess problem. Black dots form the input; squares are possible points to add that satisfy the set. A wire has two equal-cost choices, the light and dark squares, representing false and true, respectively. A variable can be represented by a wire split into a copy for each use. (Extra ends can be ended, taking care to balance the light and dark costs.) The shift gadget enables arbitrary fixing of parities and negation of wires. The crossover gadget lets wires cross. (Planar Not-All-Equal 3SAT can be solved in polynomial time.) The Not-All-Equal clause gadget has nine squares in the center (white and doubled), requiring at least $\lceil 9/2 \rceil = 5$ to be added; a total of six of these squares (all three whites) must be added if and only if all three doubled squares of the same color have been chosen by the incident wires.

References

[BCDI07] M. Bădoiu, R. Cole, E. D. Demaine, and J. Iacono. A unified access bound on comparison-based dynamic dictionaries. *Theor. Comput. Sci.*, 382(2):86–96, 2007.

[CMSS00] R. Cole, B. Mishra, J. P. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay sorting $\log n$-block sequences. *SIAM J. Comput.*, 30(1):1–43, 2000.

[Col00] R. Cole. On the dynamic finger conjecture for splay trees. Part II: The proof. *SIAM J. Comput.*, 30(1):44–85, 2000.

[DHIP07] E. D. Demaine, D. Harmon, J. Iacono, and M. Pătraşcu. Dynamic optimality—almost. *SIAM J. Comput.*, 37(1):240–251, 2007.

[DSW05] J. Derryberry, D. D. Sleator, and C. C. Wang. A lower bound framework for binary search trees with rotations. Tech. Rep. CMU-CS-05-187, Carnegie Mellon University, 2005.

[FLN03] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[Har06] D. Harmon. *New Bounds on Optimal Binary Search Trees*. PhD thesis, MIT, 2006.

[Iac01] J. Iacono. Alternatives to splay trees with $O(\log n)$ worst-case access times. In *SODA*, pages 516–522, 2001.

[Iac05] J. Iacono. Key-independent optimality. *Algorithmica*, 42(1):3–10, 2005.

[Knu71] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[Luc88] J. M. Lucas. Canonical forms for competitive binary search tree algorithms. Tech. Rep. DCS-TR-250, Rutgers University, 1988.

[Mun00] J. I. Munro. On the competitiveness of linear search. In *ESA*, pages 338–345, 2000.

[Pet08] S. Pettie. Splay trees, Davenport-Schinzel sequences, and the deque conjecture. In *SODA*, pages 1115–1124, 2008.

[ST85] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[STT86] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *STOC*, pages 122–135, 1986.

[Sun92] R. Sundar. On the deque conjecture for the splay algorithm. *Combinatorica*, 12(1):95–124, 1992.

[Tar85] R. E. Tarjan. Sequential access in play trees takes linear time. *Combinatorica*, 5(4):367–378, 1985.

[WDS06] C. C. Wang, J. Derryberry, and D. D. Sleator. $O(\log \log n)$-competitive dynamic binary search trees. In *SODA*, pages 374–383, 2006.

[Wil89] R. E. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM J. Comput.*, 18(1):56–67, 1989.