Arboral Satisfaction: Recognition and LP Approximation

Erik D. Demaine[a,1], Varun Ganesan[a], Vladislav Kontsevoi[a], Qipeng Liu[a], Quanquan Liu[a], Fermi Ma[a], Ofir Nachum[a], Aaron Sidford[a], Erik Waingarten[a], Daniel Ziegler[a]

[a] *Massachusetts Institute of Technology*

Abstract

A point set P is *arborally satisfied* if, for any pair of points with no shared coordinates, the box they span contains another point in P. At SODA 2009, Demaine, Harmon, Iacono, Kane, and Pătrascu proved a connection between the longstanding dynamic optimality conjecture about binary search trees and the problem of finding the minimum-size arborally satisfied superset of a given 2D point set [1].

We study two basic problems about arboral satisfaction. First, we develop two nontrivial algorithms to test whether a given point set is arborally satisfied. In 2D, both of our algorithms run in $O(n \log n)$ time, and one of them achieves $O(n)$ runtime if the points are presorted; we also show a matching $\Omega(n \log n)$ lower bound in the algebraic decision tree model. In d dimensions, our algorithm runs in $O(dn \log n + n \log^{d-1} n)$ time. Second, we study a natural integer linear programming formulation of finding the minimum-size arborally satisfied superset of a given 2D point set, which is equivalent to finding offline dynamically optimal binary search trees. Unfortunately, we conclude that the linear programming relaxation has large integrality gap, making it unlikely to find an approximation algorithm via this approach.

Keywords: algorithms, analysis of algorithms, computational geometry, data structures

1. Introduction

Arborally Satisfied Point Sets

The *dynamic optimality problem* asks whether there is a single binary search tree that is, up to constant factors, as good as all other binary search trees on all access sequences. This problem originated with Sleator and Tarjan's invention of splay trees [2]. They conjectured that, for any binary search tree algorithm serving a sequence X of accesses to n items at cost C, a splay tree serves the same access sequence X at cost $O(n + C)$. More generally, any binary search tree with this property is called *dynamically optimal*.

*I am corresponding author

Email address: edemaine@mit.edu (Erik D. Demaine)

Demaine, Harmon, Iacono, Kane, and Pătrascu [1] showed that dynamically optimal binary search trees are equivalent to a purely geometric problem about 2D point sets: given a point set P, find an $O(1)$-approximation to the smallest superset $P' \supseteq P$ that is *arborally satisfied* in the sense that every two points in P', not on a common horizontal or vertical line, span a rectangle containing another point in P'. The equivalence comes from viewing one dimension (x) as time and the other dimension (y) as key value. The input point set $P = (1, x_1), (2, x_2), \ldots, (m, x_m)$ represents the set $X = \langle x_1, x_2, \ldots, x_m \rangle$ of accesses. The desired point set P' represents the (key values of) nodes that are touched (visited) by the binary search tree algorithm during each access. The superset relation $P' \supseteq P$ represents that the binary search tree must in particular touch the accessed node.

Demaine et al. [1] showed that, for any binary search tree algorithm servicing access sequence X, the set P' of pairs (t, y), where node y is touched while serving the tth access x_t, is arborally satisfied. Conversely, they showed that any arborally satisfied superset $P' \supseteq P$ can be converted into a binary search tree algorithm that touches exactly the nodes at times specified by P', and thus has total cost $|P'|$. The offline optimal binary search tree is therefore equivalent to the smallest arborally satisfied superset $P' \supseteq P$, and an $O(1)$-approximation corresponds to dynamic optimality. There is also a natural notion of *online* algorithms for finding arborally satisfied supersets which corresponds (up to constant factors) to online binary search tree algorithms, but this will not be relevant here.

Our Results

We study two basic problems about arborally satisfied point sets as a combinatorial structure of fundamental importance to binary search tree data structures.

First, we develop the first nontrivial algorithms to test the arboral satisfaction property of a given point set. In Section 2, we develop a geometric algorithm resembling a sweep line which runs in $O(n \log n)$ time. In Section 3, we give an alternate $O(n \log n)$ algorithm that relies on the correspondence between arboral satisfaction and binary search tree algorithms. This algorithm runs in $O(n)$ time if the points are presorted. In Section 4, we give a matching $\Omega(n \log n)$ lower bound on testing arboral satisfaction in the algebraic decision tree model, by a reduction from set equality. In Section 5, we consider an extension of arboral satisfaction to d dimensions for $d > 2$, and give an $O(dn \log n + n \log^{d-1} n)$-time algorithm for testing arboral satisfaction.

Second, we consider a linear programming relaxation approach to approximating the minimum-size arborally satisfied superset (offline optimal binary search tree). The current best polynomial-time approximation algorithm achieves an approximation ratio of $O(\log \log n)$ [3]; for dynamic optimality, the goal is to obtain a constant-factor approximation algorithm. In Section 6, we formulate two "natural" integer linear programming formulations of the problem. In Section 6.3, we show that the two linear program relaxations are unlikely to yield constant-factor approximations, as both exhibit an integrality gap of $\Omega(\log n)$ even in the average case.

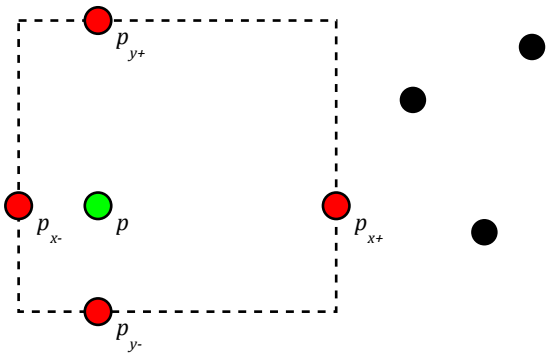

Figure 1: An image of R_p

2. A Geometric Algorithm for Testing Arboral Satisfaction

We define the x-predecessor of a point $p = (x_p, y_p) \in P$ to be first point that the ray from p to $(-\infty, y_p)$ encounters, and we denote it by p_{x-}. If there is no such point, then $p_{x-} = (-\infty, y_p)$. Similarly, the x-successor of a point p is the first point that the ray from p to (∞, y_p) encounters, and we denote it by p_{x+}. Again, if there is no such point, we set $p_{x+} = (\infty, y_p)$. The y-predecessor and y-successor are defined and denoted analogously.

Given two points p and q with distinct x and y coordinates, the *induced box* of p and q is the unique axis-aligned rectangle with p and q as corners. If the point set P is not arborally satisfied, then there exist two points p and q with an induced box containing no additional points.

Given a point set P, we first sort the points by x-coordinate and store the points in an array A, breaking ties by y-coordinate. We also store the points in another array T ordered by the y-coordinate, breaking ties by x-coordinate. Finally, we build a layered range tree L on P to answer orthogonal range queries [4].

For each point $p \in P$, let R_p be the unique rectangle defined by the points p_{x-}, p_{x+}, p_{y-}, and p_{y+}. An example of a rectangle is shown in Figure 1.

2.1. Algorithm.

The algorithm works by sweeping through all points in A. For each point, we run an orthogonal range query on L with the strict interior of R_p as the query object. If it contains a point other than p, the algorithm returns `False` and terminates. Otherwise, it moves on to the next point. If the algorithm processes every point, it returns `True`.

2.2. Correctness and Runtime Analysis.

Theorem 2.1 shows that the algorithm is correct.

Theorem 2.1. *There exists $p \in P$ such that R_p contains a point other than p if and only if P is not arborally satisfied.*

The theorem follows directly from Lemma 2.2 and Lemma 2.3; Lemma 2.2 shows that if the set is not arborally satisfied, the algorithm returns `False`, and Lemma 2.3 shows that if the algorithm returns `False`, the points are not arborally satisfied.

Lemma 2.2. *Suppose points p and q differ in both x and y coordinates and their induced box does not contain any other point in its interior. Also, assume the coordinates of q are greater than the corresponding coordinates of p. Then q is contained in the interior of R_p.*

Proof. Suppose for the sake of contradiction that q is not contained in the interior of R_p. Then for some coordinate k, p_{k+}, the successor of p in the k-coordinate, has $(p_{k+})_k \leq (q)_k$, where $(p_{k+})_k$ and $(q)_k$ denote the k-coordinates of p_{k+} and q, respectively. Since p_{k+} and p have the same value in the other coordinate, this implies that p_{k+} is contained within the box induced by p and q, a contradiction. □

Lemma 2.3. *If R_p contains a point other than p in its interior, then the point set is not arborally satisfied.*

Proof. Of the points contained in R_p, let q be the closest one to p in the Euclidean distance metric. q must differ from p in both x and y coordinates in order for it not to be a successor or a predecessor of p. The box induced by p and q does not contain any points in its interior, because any point strictly inside this box would be strictly within R_p and closer to p, a contradiction. □

It follows that the algorithm as stated is correct. It takes $O(n \log n)$ to sort the points and arrange them in array A and array T. It also takes $O(n \log n)$ to build the layered range tree R. It takes $O(1)$ to find predecessors and successors in T and A; therefore, it takes $O(1)$ to build R_p from a point p. Additionally, it takes $O(\log n)$ to query R. Since the algorithm must build R_p and query R for each point p, the algorithm takes $O(n \log n)$ time.

3. Treap Algorithm for Testing Arboral Satisfaction

We can also check for arboral satisfaction by using the correspondence between arborally satisfied point sets and valid BST executions.

Consider the function f which maps valid BST executions E to points in the plane by placing a point (x, i) for each access to value x and time i. Then, given an arborally satisfied point set P, there exists a BST execution E such that $f(E) = P$ [1].

To construct this BST execution, we define the next access time $N(x, i)$ of x at time i to be the first y coordinate of any point in P encountered by the ray from (x, i) to (x, ∞). If no point is encountered, $N(x, i) = \infty$. The BST will be a Cartesian tree (treap) on the points $(x, N(x, i))$, which is a BST on the first coordinate and a min-heap on the second coordinate [5].

Let τ_i denote the set of points with second coordinate i. At time i, the BST execution touches all points in τ_i and then rotates those points to preserve the treap order on those points for time $i + 1$. Demaine et al. show that this is sufficient to guarantee the entire treap is in treap order at the next step [1]. They also show that if and only if the original set of points is arborally satisfied, τ_i will form a continuous subtree at the root at each time step i [1, Lemma 2.1, Lemma 2.2].

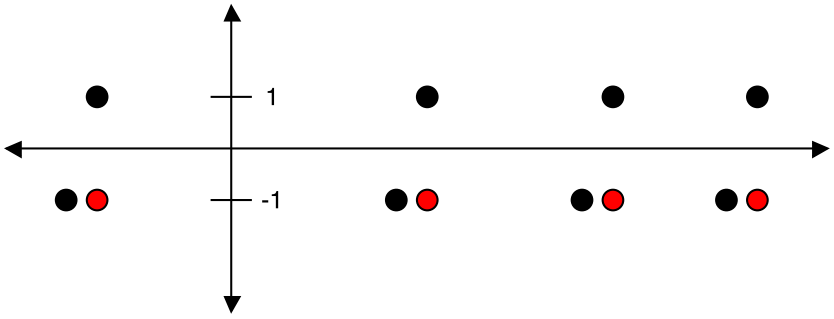

Figure 2: Visual representation of the set equality construction. Points in S correspond to black points, and points in T correspond to red points.

3.1. Algorithm.

The arboral satisfaction testing algorithm works by building the treap and performing rotations at each time step i on τ_i to preserve treap order. If, at any time step i in the BST execution, the points τ_i do not form a continuous subtree at the root, the algorithm returns `False`. Otherwise, if the BST execution completes, the algorithm returns `True`.

3.2. Runtime Analysis

After having sorted the input point set, the work done by this algorithm is wholly treap-related. Construction of the treap may be done in linear time via a Cartesion Tree construction [6]. When the algorithm processes n_i points at time i, the n_i nodes form a connected sub-tree at the root of the treap (otherwise the algorithm terminates). It takes $O(n_i)$ time to update the next touch times and at most 2 rotations for each node in this subtree, yielding an $O(n_i)$ running time. Therefore, since $\sum n_i = |P|$, all treap operations in total aggregate to a linear running time.

Excluding initial sorting, this is a linear time algorithm for determining arboral satisfaction. When sorting is necessary, the running time becomes $O(n \log n)$.

4. An $\Omega(n \log n)$ Lower Bound

Theorem 4.1 (Testing Arboral Satisfaction Lower Bound)**.** *Any deterministic arboral satisfaction testing algorithm A in the algebraic decision tree model must have running time $\Omega(n \log n)$.*

Proof. We reduce from the set equality problem: Given two sets S and T, it takes $\Omega(n \log n)$ time to determine if they are equal.

First, we check if $|S| = |T|$. If so, the following procedure checks that $S \subseteq T$, and hence that $S = T$. For each $s \in S$, draw the two points $(s, 1)$ and $(s - \epsilon, -1)$ where ϵ is very small. For each $t \in T$, draw the point $(t, -1)$. See Figure 2 for an example. This arrangement of points is arborally satisfied if and only if S and T are the same set, so checking for arboral satisfaction is $\Omega(n \log n)$.

□

5. Higher Dimensions

We can define arboral satisfaction in d-dimensions analogously. A point set with n points is *arborally satisfied* if, for any two points which differ in at least two coordinates, their induced box contains at least one other point. This means that for any two points, there is a monotone path that goes through other points in orthogonal directions.

5.1. d Dimensions.

The algorithm is identical to the two-dimensional algorithm, except that we query the rectangle defined by the $2d$ points p_{x_i-} and p_{x_i+} (for all i).

5.2. Implementation Details and Runtime Analysis.

We use $x_1, x_2, \ldots, x_d$ to denote the coordinates of the space. Sorting the point set P d times to form d sorted arrays takes time $O(dn \log n)$ if we proceed as follows:

- Use a stable sort such as merge-sort to sort first by x_d, then by x_{d-1}, and in all coordinates in order. In $O(dn \log n)$ time, this yields a lexicographically sorted array, primarily sorted by x_1, secondarily sorted by x_2, and so on, with x_d being the least significant key.
- Make a copy of this array. Sort by x_d. In $O(n \log n)$ time, this yields an array primarily sorted by x_d, secondarily sorted by x_1, and so on, with x_{d-1} being the least significant key.
- Repeat this process $d-2$ times (copying and sorting by x_{d-1}, then by x_{d-2}, and so on) so that each of our coordinates in turn is the least significant coordinate in a sorted array.

For any $p \in P$, we can build the rectangle R_p in $O(d \log n)$ by finding predecessors and successors in each coordinate with the sorted lists.

Building the layered range tree in d dimensions can be done in $O(n \log^{d-1} n)$ time. Each query to the layered range tree will take $O(\log^{d-1} n)$ time. Therefore, the total time for n queries is $O(n \log^{d-1} n)$.

The total running time is $O(dn \log n + n \log^{d-1} n)$.

6. Equivalent Integer Linear Programs

We consider two integer linear programming relaxations of this problem.

It is natural to restrict attention to the grid of intersection points "induced" by the set of points P: extend horizontal and vertical lines through each point in P and consider only the $O(n^2)$ intersections of these lines. It is easy to show that in any arborally satisfied point set P, points that are not grid intersections can be moved to positions that are.

We can directly correspond an integer linear program to the problem by assigning an indicator variable b_{jk} to each grid point, where the variable is set to 1 if the point is included and is 0 otherwise, and j and k are labels on the grid intersection points (labelled so that

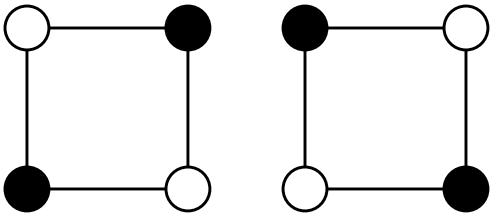

Figure 3: Positive and negative rectangles: black corresponds to points that are included, and white corresponds to the points not included

b_{00} is the lower left grid point, b_{01} is the point directly above, and b_{10} is the point directly to the right).

This immediately gives us n constraints, one for each variable corresponding to a point in P (setting them all equal to 1). The linear programming objective is to minimize the sum of all the variables; the task that remains capturing the arboral satisfaction property with further constraints.

6.1. Quadratic Constraint Size

Consider the following set of constraints:

$$2b_{ij} + 2b_{nm} - \left(\sum_{i \le k \le n} \sum_{j \le l \le m} b_{kl} \right) \le 1 \qquad \forall i < n, j < m \tag{1}$$

$$2b_{im} + 2b_{nj} - \left(\sum_{i \le k \le n} \sum_{j \le l \le m} b_{kl} \right) \le 1 \qquad \forall i < n, j < m \tag{2}$$

These constraints exactly capture arboral satisfaction. Constraints of type (1) correspond to all possible "positive" rectangles, and constraints of type (2) correspond to all possible "negative" rectangles (explained in Figure 2).

To see the equivalence, consider constraints of type (1). If points in the grid corresponding to b_{ij} and b_{nm} are present, the rectangle they span must be satisfied by some point corresponding to b_{kl} such that $i \le k \le n$ and $j \le l \le m$ (although it cannot equal to b_{ij} or b_{nm}). The constraint ensures precisely this: if $b_{ij} = b_{nm} = 1$, at least one other b_{kl} in the parenthesized term must be set to 1, or else the entire left hand side will be a quantity greater than 1. Constraints of type (2) work the same way.

6.2. Linear Constraint Size

By definition, arborally satisfied point sets have all rectangles satisfied by some point on the interior (including edges). It turns out that a short inductive argument shows that all rectangles are in fact satisfied by some point on their edges [1]. Thus, we can reduce the constraint size by "only" requiring that rectangles be satisfied on their edges. This gives the following set of constraints:

$$b_{ij} + b_{nm} - \left(\sum_{l=i+1}^{n-1} (b_{lj} + b_{lm}) + \sum_{l=j+1}^{m-1} (b_{il} + b_{nl}) + b_{im} + b_{nj} \right) \leq \quad 1 \qquad \forall i < n, j < m \quad (3)$$

$$b_{im} + b_{nj} - \left(\sum_{l=i+1}^{n-1} (b_{lj} + b_{lm}) + \sum_{l=j+1}^{m-1} (b_{il} + b_{nl}) + b_{ij} + b_{nm} \right) \leq \quad 1 \qquad \forall i < n, j < m \quad (4)$$

Constraints of type (3) correspond to all possible "positive" rectangles, and constraints of type (4) correspond to all possible "negative" rectangles. This captures arboral satisfaction for the same reason that the constraints of types (1) and (2) do. The only difference is that we no longer include points that are strictly inside rectangles.

6.3. Unbounded Integrality Gap

Unfortunately, in any of these instances, we can satisfy the relaxed linear program by setting all variables corresponding to neighboring points of points in P to 0.5. Refer to Figure 4 for the setup: black circles are variables set to 1, red circles are variables set to 0.5, and white circles representing empty spots are set to 0).

This solution turns out to always be feasible. Note that every constraint corresponding to rectangles spanned by points in P will be satisfied, since they contain at least 2 points of value 0.5. All constraints corresponding to rectangles where one corner is set to 1 and the opposite corner is set to 0.5 will satisfied, as at least one other variable of value 0.5 (adjacent to the corner set to 1) will be subtracted away. Finally each constraint corresponding to a rectangle with opposite corners set to 0.5 will automatically be satisfied. This solution simply adds at most 2 to the sum of all the variables for each point in P, and is therefore $O(n)$.

It is known that some sets of n points require the addition of $\Omega(n \log n)$ additional points to be arborally satisfied: e.g., the bit-reversal sequence and random sequences in expectation [7, 1]. This implies an $\Omega(\log n)$ integrality gap for both linear programs, even in the average case.

7. Conclusion

We are hopeful that our study of arborally satisfied sets will ultimately help resolve the dynamic optimality problem. In particular, our optimal algorithms for testing arboral satisfaction may give some structural insights.

The major open problem left by our work is to achieve a constant-factor approximation to the minimum-size arborally satisfied superset. We have shown one approach to this problem that is unlikely to work, enabling a more focused search in the future.

Another interesting question is whether there is a data structural (or other) application to our higher-dimensional generalization of arborally satisfied point sets.

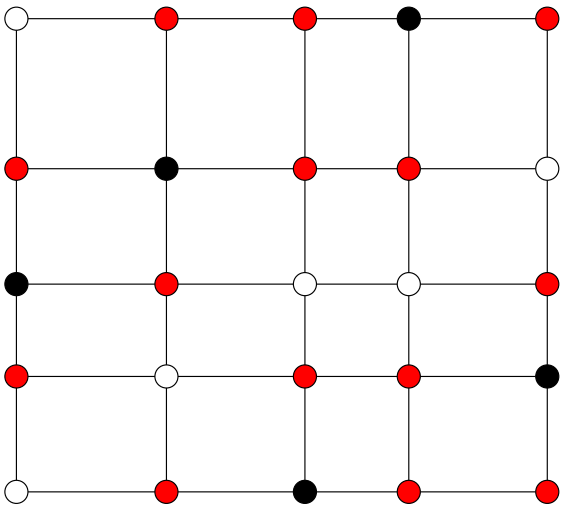

Figure 4: Integrality gap of the integer program. Black circles represent points in P given weight 1, red circles represent points included with weight $\frac{1}{2}$, and white circles are not included and given weight 0.

Acknowledgements

This work began in an open-problem solving session for MIT's Advanced Data Structures class (6.851) in Spring 2014. We thank any unnamed participants of the session for contributing to this solution.

E. Demaine's work is supported in part by MADALGO — Center for Massive Data Algorithmics — a Center of the Danish National Research Foundation.

References

[1] E. D. Demaine, D. Harmon, J. Iacono, D. Kane, M. Pătraşcu, The geometry of binary search trees, in: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, New York, New York, 2009, pp. 496–505.
URL http://dl.acm.org/citation.cfm?id=1496770.1496825

[2] D. D. Sleator, R. E. Tarjan, Self-adjusting binary search trees, J. ACM 32 (3) (1985) 652–686. doi:10.1145/3828.3835.
URL http://doi.acm.org/10.1145/3828.3835

[3] E. D. Demaine, D. Harmon, J. Iacono, M. Pătraşcu, Dynamic optimality—almost, SIAM Journal on Computing 37 (1) (2007) 240–251.

[4] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars", "orthogonal range searching", in: "Computational Geometry: Algorithms and Applications", "3rd" Edition, "Springer", "2008", Ch. 5, pp. "95–120".

[5] J. Vuillemin, A unifying look at data structures, Commun. ACM 23 (4) (1980) 229–239. doi:10.1145/358841.358852.
URL http://doi.acm.org/10.1145/358841.358852

[6] D. Harel, R. E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM Journal on Computing 13 (2) (1984) 338–355.

[7] R. Wilber, Lower bounds for accessing binary search trees with rotations, SIAM Journal on Computing 18 (1) (1989) 56–67.