The Diameter of Caterpillar Associahedra

Benjamin Aram Berendsohn
Institut für Informatik, Freie Universität Berlin, Germany

Abstract

The caterpillar associahedron $\mathcal{A}(G)$ is a polytope arising from the rotation graph of search trees on a caterpillar tree G, generalizing the rotation graph of binary search trees (BSTs) and thus the conventional associahedron. We show that the diameter of $\mathcal{A}(G)$ is $\Theta(n + m \cdot (H + 1))$, where n is the number of vertices, m is the number of leaves, and H is the entropy of the leaf distribution of G.

Our proofs reveal a strong connection between caterpillar associahedra and searching in BSTs. We prove the lower bound using Wilber's first lower bound for dynamic BSTs, and the upper bound by reducing the problem to searching in static BSTs.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Theory of computation → Data structures design and analysis

Keywords and phrases Graph Associahedra, Binary Search Trees, Elimination Trees

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.14

Funding Work supported by DFG grant KO 6140/1-1.

Acknowledgements I would like to thank László Kozma for helpful discussions and suggestions.

1 Introduction

Associahedra are a family of polytopes with interesting combinatorial properties. In particular, the skeleton of the $(n-1)$-dimensional associahedron $\mathcal{A}_n$ represents the *rotation graph* of binary search trees (BSTs) on n keys. More precisely, each vertex of $\mathcal{A}_n$ represents a BST, and each edge represents a rotation (definitions of BSTs and rotations can be found in standard textbooks). The diameter of $\mathcal{A}_n$ is known to be precisely $2n-6$ when $n > 10$ [21, 19]; that is, every BST can be transformed into any other BST with at most $2n-6$ rotations, and this is tight.

In this paper, we study a generalization of associahedra called *graph associahedra*. Graph associahedra were originally defined using *tubings* [6]. We use an equivalent definition based on *search trees on graphs* (STGs). While the keyspace of a BST is a linearly ordered set, the key space of an STG is a graph. Formally, given a connected graph $G = (V, E)$, a *search tree on* G is a rooted tree T that can be constructed as follows. Choose a vertex $r \in V$ as the root. Then, recursively create search trees on the connected components of $G \setminus v$, and add them to T as children of r. Rotations on STGs can be defined similarly as for BSTs (more details in Section 2). Search trees on graphs have been used in various contexts under different names (see, e.g., [3, Section 2.2]).

Given a connected graph G on n vertices, Carr and Devadoss [6] defined the graph associahedron $\mathcal{A}(G)$ as an $(n-1)$-dimensional polytope such that the skeleton of $\mathcal{A}(G)$ is isomorphic to the rotation graph of the search trees on G. Since search trees on the path with n vertices correspond to BSTs on n nodes, we obtain the conventional associahedron when G is a path.

For search trees T_1 and T_2 on a graph G, let the *rotation distance* $d(T_1, T_2)$ be the minimum number of rotations required to transform T_1 into T_2. The diameter $\delta(\mathcal{A}(G))$ of $\mathcal{A}(G)$ is the maximum rotation distance between two search trees on G. Manneville and Pilaud [15] showed that the diameter of graph associahedra is monotone under the addition of edges, and $\max\{2n-18, m\} \leq \delta(\mathcal{A}(G)) \leq \binom{n}{2}$ holds for each connected graph G on n

vertices and m edges. Both the upper and the lower bound are asymptotically tight: For example, conventional associahedra (G is a path) and cyclohedra (G is a cycle) have linear diameter, and permutohedra (G is a complete graph) have diameter $\binom{n}{2}$.

In this paper, we consider the case where G is a *caterpillar tree*. A caterpillar tree (or simply *caterpillar*) is a tree consisting of a path and some number of leaves that are adjacent to the path. The choice of that path is not unique for non-trivial caterpillars (it is always possible to either add or remove a leaf at the end of the path), but we assume that any considered caterpillar consists of a distinguished path called the *spine* and any number of leaves, called *legs*. Our asymptotic results are not affected by the choice of the spine.

We determine the diameter of every caterpillar associahedron up to a constant factor. This involves the Shannon entropy of the "leg distribution", which we now properly define. Let G be a caterpillar tree with n spine vertices $s_1, s_2, \ldots, s_n$, let s_i be adjacent to m_i leg vertices, and let $m = m_1 + m_2 + \cdots + m_n$ be the total number of leg vertices. Then

$$H(G) = H(m_1, m_2, \ldots, m_n) = \sum_{i \in [n], m_i > 0} \frac{m_i}{m} \log\left(\frac{m}{m_i}\right).$$

For simplicity of presentation, we write $H'(\cdot) = H(\cdot) + 1$. We are now ready to state our main result.

▶ **Theorem 1.1.** *Let G be a caterpillar tree with n spine vertices and m leg vertices. Then $\delta(\mathcal{A}(G)) \in \Theta(n + m \cdot H'(G))$.*

Notably, if $m = n$ and each spine node is adjacent to one leaf node, then $\delta(\mathcal{A}(G)) \in \Omega(n \log n)$.

Our proofs make use of techniques from the design of *optimal BSTs*. A connection between rotations in BSTs and rotations in search trees on caterpillars is not surprising – caterpillars are similar to paths, after all. However, we show a connection to *queries* to BSTs. Essentially, the leg nodes in search trees on caterpillars can be seen as queries to the BST on the spine nodes. For our upper bound (Section 4), we use the fact that an optimal *static* BST for an input distribution X has amortized query cost $H'(X)$ [16]. For our lower bound (Section 5), we use *Wilber's first lower bound* [22], which bounds the performance of *dynamic* BSTs on a certain input sequence. We show that it also bounds the rotation distance between certain search trees on a caterpillar. Finally, we show that Wilber's first lower bound is asymptotically equal to $H'(X)$ if the input distribution X is fixed, but the order of queries is worst possible. Note that this implies that dynamic BSTs cannot beat optimal static BST on any distribution if the ordering is worst possible. Kujala and Elomaa [12] previously showed that this is true even if the ordering is random, but they did not use Wilber's bound.

1.1 Related work

Improved bounds on $\delta(\mathcal{A}(G))$ are known if G belongs to certain graph classes. Pournin [18] showed that $\delta(\mathcal{A}(G)) \approx 2.5n$ if G is the cycle on n vertices. Manneville and Pilaud [15] noted that $\delta(\mathcal{A}(G)) \in \mathcal{O}(n \log n)$ if G is a tree on n vertices, and Cardinal, Langerman and Pérez-Lantero [3] showed this bound is tight if G has the form of a balanced binary tree.

Recently, Cardinal, Pournin, and Valencia-Pabon [4, 5] showed that $\delta(\mathcal{A}(G)) \in \mathcal{O}(\mathrm{td}(G) \cdot n)$, where $\mathrm{td}(G)$ is the *treedepth*[1] of G, and that this bound is attained by *trivially perfect graphs*. Using the relationship between treedepth and *treewidth*, this extends the $\mathcal{O}(n \log n)$

[1] The treedepth $\mathrm{td}(G)$ can be defined as the minimum height of a search tree on G.

upper bound to graphs with bounded treewidth. They also showed that this bound is is tight for graphs of *pathwidth two* (which have treewidth at most two, but are not necessarily trees). For the definitions of treewidth and pathwidth, we refer to [4]. Our Theorem 1.1 shows that the $\mathcal{O}(n \log n)$ bound is tight already for caterpillars, which are both trees and have pathwidth *one* (in fact, caterpillars are precisely the graphs of pathwidth one).

We do not consider *queries* to STGs in this paper. Some results from BSTs have been shown to hold in the more general case where G is a tree. Bose, Cardinal, Iacono, Koumoutsos, and Langerman [2] presented an $\mathcal{O}(\log \log n)$-competitive search tree algorithm based on *tango trees* for BSTs [10]. Berendsohn and Kozma [1] described a variant of Splay trees [20], and a polynomial-time approximation scheme for the optimal static search tree on a given tree for a given input distribution. Notably, it is still unknown whether an optimal static search tree on a tree can be found in polynomial time.

Berendsohn and Kozma [1] also showed that if we only consider a subset of search trees on a tree G called k-cut trees, then the maximum rotation distance between two STGs is linear. A special case of k-cut trees are *Steiner-closed trees*, which play a central role in the results of [2] and [1].

2 Preliminaries

In this paper, we consider (simple and undirected) graphs on the one hand, and (rooted) search trees on the other. We call the vertices of search trees *nodes*. In both cases, we denote by $V(\cdot)$ the set of vertices or nodes and by $E(\cdot)$ the set of edges.

Let G be a graph. We denote the subgraph of G induced by $U \subseteq V(G)$ by $G[U]$. For $v \in V(G)$, we write $G \setminus v = G[V(G) \setminus \{v\}]$.

Let T be a rooted tree and $x \in V(T)$. For a node x, T_x denotes the subtree of T consisting of x and all its descendants. The *depth* of x is the number of nodes in the path from the root of T to x, and is denoted by $\operatorname{depth}_T(x)$.

2.1 Queries in binary search trees

In the *dynamic BST model*, we are given a starting BST S on $[n]$ and a sequence σ of access queries. Each access query specifies a node $i \in [n]$. We start each query with a pointer at the root, and are required to move the pointer to the node i to satisfy the query. To this end, we are allowed to move the pointer to the the parent or a child of the node it is currently pointing at, or execute a rotation involving that node. Let $\operatorname{OPT}(S, \sigma)$ denote the minimum number of pointer moves and rotations needed to serve σ. We charge a pointer move at the start of each query, when the pointer is moved to the root, so each query has cost at least one.

Since the rotation distance between two BSTs is $\mathcal{O}(n)$, we can always replace the starting BST S by a different one at the cost of $\mathcal{O}(n)$. If the access sequence is long enough, this cost is insignificant; therefore, let us define $\operatorname{OPT}(\sigma) = \min_S \operatorname{OPT}(S, \sigma)$. For each BST S, we have $\operatorname{OPT}(S, \sigma) \leq \operatorname{OPT}(\sigma) + \mathcal{O}(n)$.

It is not known how to compute or approximate $\operatorname{OPT}(S, \sigma)$ or the associated sequence of operations efficiently. However, a number of algorithms have been conjectured to be *instance-optimal*, i.e, to serve every access sequence σ with a cost of $\mathcal{O}(\operatorname{OPT}(\sigma) + n)$, most notably *Splay* [20] and *Greedy* [14, 17]. We emphasize that Splay is an *online* algorithm, i.e., it serves each query independently from future queries, and that Greedy can be made online [9] with only a constant-factor overhead. It is currently unknown whether any online algorithm can approximate the offline optimum $\operatorname{OPT}(\sigma)$ by a constant factor; this is the subject of the *dynamic optimality conjecture*.

There are several lower bounds known for $\mathrm{OPT}(\sigma)$. In this paper, we use *Wilber's first lower bound* [22], which we define and discuss in Section 5.1.

If we do not allow rotations, then the best strategy is to simply move the pointer down until we hit the queried node. Thus, the minimum cost of serving $\sigma = (x_1, x_2, \ldots, x_m)$ in S is $\sum_{i=1}^{m} \mathrm{depth}_S(x_i)$. A BST S minimizing this quantity is called an *optimal static BST*. Note that only the frequencies of the elements in σ affect the static cost, not the order. For a BST S on $[n]$ and element frequencies $m_1, m_2, \ldots, m_n \in \mathbb{N}_0$, define

$$\mathrm{cost}(S, m_1, m_2, \ldots, m_n) = \sum_{i=1}^{n} m_i \cdot \mathrm{depth}_S(i).$$

Let $\text{OPT-ST}(m_1, m_2, \ldots, m_n)$ be the minimum $\mathrm{cost}(S, m_1, m_2, \ldots, m_n)$ over all possible BSTs S.

It is possible to compute an optimal static BST in $\mathcal{O}(n^2)$ time [11]. Mehlhorn showed that $\frac{1}{m}\text{OPT-ST}(m_1, m_2, \ldots, m_n)$ is close to the entropy of the input distribution.

▶ **Lemma 2.1** (Mehlhorn [16]). *Let $X = (m_1, m_2, \ldots, m_n)$ be a sequence of nonnegative integers, and let $m = m_1 + m_2 + \cdots + m_n$. Then*

$$\frac{1}{2}H(X) \cdot m \leq \textit{OPT-ST}(X) \leq (2H(X) + 2) \cdot m = 2H'(X) \cdot m.$$

2.2 Search trees on graphs

Let G be a connected graph, and T be a rooted tree, such that $V(T) = V(G)$. Let r be root of T and let $c_1, c_2, \ldots, c_k$ be the children of r in T. Then T is a *search tree on G* if

(a) $G \setminus r$ consists of precisely k connected components $C_1, C_2, \ldots, C_k$ such that $V(T_{c_i}) = V(C_i)$ for each $i \in [k]$; and

(b) T_{c_i} is a search tree on C_i for each $i \in [k]$.

Let T be a search tree on G, let $p \in V(G)$, let c be a child of p in T, and let g be the parent of p in T, if p is not the root. A *rotation* of the edge (p, c) makes c the parent of p and child of g (or root), and accordingly redistributes children so that the result is still a search tree on G. More precisely, it (1) makes c a child of g, if p is not the root, and otherwise, makes c the root; (2) makes p a child of c; and (3) makes each child x of c a child of p where $V(T_c)$ contains both a vertex adjacent to p and a vertex adjacent to c. See Figure 1 for an illustration. It can be checked that the rooted tree resulting from a rotation is indeed a search tree on G. It is also easy to see that each search tree on G can be rotated into every other search tree on G, e.g., by rotating the correct element to the root and then recursing on the subtrees.

2.3 Projections of STGs

Both of our proofs make use of a concept defined by Cardinal, Langerman, and Pérez-Lantero [3]. Let T be a search tree on a graph G, and let v be a leaf vertex in G. Note that v has at most one child in T. We define $T \setminus v$ to be the following search tree on $G \setminus v$: If v has no children, simply remove v. If v has a parent p and a child c, remove v and make c a child of p. If v is the root of T and has a child c, then remove v and make c the root. We call this operation *pruning* v.

If G is a tree, then we can obtain every subgraph of G by progressively removing leaves. Accordingly, if T is a search tree on a tree G, and $U \subseteq V(G)$ is a set of vertices such that $G[U]$ is connected, then we can define the *projection* of T onto U, written $T[U]$, as the search

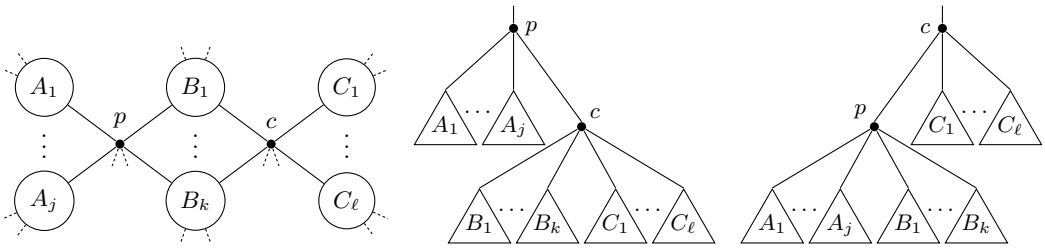

Figure 1 A rotation in a search tree on G. *(left)* A subgraph G' of G with two vertices p and c that split G' into $j + k + \ell$ components. Each solid line represents one or more edges. Dashed lines indicate possible edges to the rest of G. *(center)* A subtree T_p in a search tree T on G, with $V(T_p) = V(G')$. *(right)* The result of the rotation (p, c) in T.

tree on $G[U]$ obtained by progressively pruning the vertices in $V(G) \setminus U$. It is easy to see that the order of pruning does not matter.

The main utility of projections lies in the following lemma, which essentially states that projections onto U are only affected by rotations between nodes in U.

▶ **Lemma 2.2** (Cardinal et al. [3]). *Let T be a search tree on a tree G, let $U \subseteq V(G)$ such that $G[U]$ is connected, let (x, y) be an edge of T, and let T' be the tree obtained by rotating (x, y). If $x, y \in U$, then $T'[U]$ is the STG obtained when rotating (x, y) in $T[U]$. Otherwise, $T'[U] = T[U]$.*

3 Search trees on caterpillars

Let $n \in \mathbb{N}_+$ and $m_1, m_2, \ldots, m_n \in \mathbb{N}_0$, and write $m = \sum_{i=1}^{n} m_i$. We define the caterpillar $C(m_1, m_2, \ldots, m_n)$ with n spine vertices and m leg vertices as follows. The spine of G consists of the vertices $s_1, s_2, \ldots, s_n$, in that order. Additionally, for each $i \in [n]$ and $j \in [m_i]$, there is a leg vertex $\ell_{i,j}$ that is adjacent to s_i. Clearly, every caterpillar can be constructed this way.

Let T be a search tree on $G = C(m_1, m_2, \ldots, m_n)$, and let $x \in V(T)$. We call x a *leg node* if it corresponds to a leg vertex, and a *spine node* if it corresponds to a spine vertex. We denote nodes in T in the same way we denote vertices in G, i.e., we write $\ell_{i,j}$ for leg nodes and s_i for spine nodes.

We call a leg node *bound* if it has no children, and *free* otherwise.

▶ **Observation 3.1.** *Let T be a search tree on $C(m_1, m_2, \ldots, m_n)$. Consider a leg node $\ell_{i,j}$. If $\ell_{i,j}$ has no children, then s_i is its parent. Otherwise, $\ell_{i,j}$ has exactly one child, and s_i is a descendant of $\ell_{i,j}$.*

Define $\mathrm{bst}(T)$, the *spine BST*, as the projection of T onto the spine vertices of G (see Figure 2 for an example). Since the spine vertices form a path, $\mathrm{bst}(T)$ indeed corresponds to a binary search tree. By Lemma 2.2, each rotation between two spine nodes in T corresponds to a BST rotation in $\mathrm{bst}(T)$. However, the converse is not true in general, since two nodes u, v that are parent and child in $\mathrm{bst}(T)$ might have leg nodes between them in T, in which case a rotation of u, v in $\mathrm{bst}(T)$ cannot be applied to T. Call an edge (u, v) of $\mathrm{bst}(T)$ *light* if (u, v) is also an edge in T. Essentially, as long as we restrict ourselves to light edges, we can apply BST restructuring algorithms to T. This will be useful to prove our upper bound. We further observe that rotations between spine nodes preserve the parents of leg nodes.

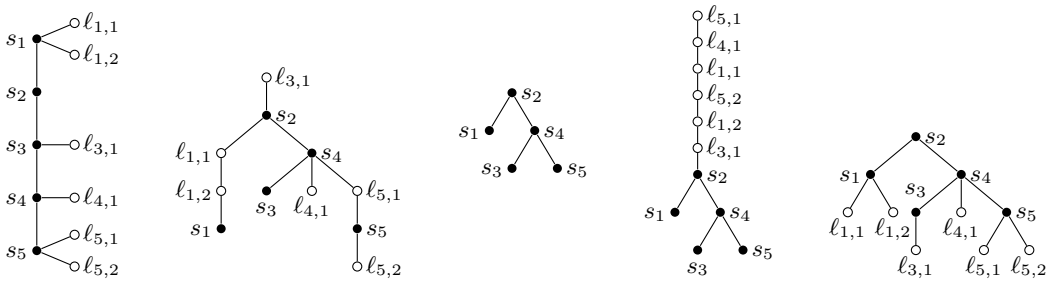

Figure 2 *(left)* The caterpillar $G = C(2, 0, 1, 1, 2)$. *(center left)* A search tree T on G. *(center)* The spine BST $S = \text{bst}(T)$. *(center right)* The STG $A(S, \pi)$ with $\pi = (\ell_{3,1}, \ell_{1,2}, \ell_{5,2}, \ell_{1,1}, \ell_{4,1}, \ell_{5,1})$. *(right)* The STG $B(S)$.

▶ **Observation 3.2.** *Let T be a search tree on a caterpillar, let T' arise from a rotation between spine nodes in T, and let ℓ be a leg node. If ℓ is the root of T, then ℓ is also the root of T'. Otherwise, the parent of ℓ in T' is equal to the parent of ℓ in T.*

3.1 Special STGs

Before proceeding with the proofs, we define two useful kinds of STGs where the spine BST has only light edges. Let $G = C(m_1, m_2, \ldots, m_n)$ be a caterpillar, let S be a BST on the spine nodes of G, and let π be an ordering of the leg nodes of G. Define $A(S, \pi)$ to be the unique search tree on G such that $\text{bst}(A(S, \pi)) = S$, each leg node is above each spine node (i.e., the leg nodes form a path at the top of the tree), and the order of the leg nodes from *bottom to top* is π. Define $B(S)$ to be the unique search tree on G such that all leg nodes are bound and $\text{bst}(B(S)) = S$. Clearly, every search tree T on G without free leg nodes is equal to $B(\text{bst}(T))$. Figure 2 shows examples.

4 Upper bound

Fix a caterpillar $G = C(m_1, m_2, \ldots, m_n)$. We first consider only STGs without free leg nodes.

▶ **Lemma 4.1.** *Let T_1, T_2 be search trees on G without free leg nodes. Then, T_1 can be transformed into T_2 with $\mathcal{O}(n)$ rotations.*

Proof. Let $S_1 = \text{bst}(T_1)$ and $S_2 = \text{bst}(T_2)$. As stated in the introduction, there is a sequence of at most $\mathcal{O}(n)$ rotations that transforms S_1 into S_2. We simply apply these rotations to T_1. For this to be well-defined, we need to show that we never (attempt to) rotate a heavy edge. Since T_1 has no free leg nodes, all edges in S_1 are light, so the first rotation goes through. Furthermore, a rotation between two spine nodes can never change a leg node from bound to free (or vice versa). Thus, by induction, after each rotation, all leg nodes are still bound, so we can apply the next rotation. ◀

The above lemma provides us with a "core" of the caterpillar associahedron with linear diameter. In the following, we show that the rotation distance from any search tree to *some* STG without free leg nodes is $\mathcal{O}(n + H'(G) \cdot m)$. By the triangle inequality, this means that the rotation distance between any two STGs is at most $2 \cdot \mathcal{O}(n + H'(G) \cdot m) + \mathcal{O}(n) = \mathcal{O}(n + H'(G) \cdot m)$, and thus we have the upper bound of Theorem 1.1.

We first show how to reduce the problem to the case that $T = A(S, \pi)$ for some BST S and some leg node ordering π. For this, the following lemma is useful.

▶ **Lemma 4.2.** *Let T be a BST. There exists a sequence of $\mathcal{O}(n)$ rotations on T such that*
(i) *every rotation involves only nodes at depth at most 3; and*
(ii) *every node becomes the root of* $\mathrm{bst}(T)$ *at some point.*

Lemma 4.2 can be easily derived from a result by Cleary [8], which uses algebraic techniques. We provide an elementary proof for completeness.

Proof of Lemma 4.2. We proceed in two phases. In the first phase, we repeatedly rotate the root of T with its left child, until the root has no left child. This requires $\mathcal{O}(n)$ rotations.

In the second phase, we repeat the following step as long as the root has a right child u. If u has a left child v, then rotate u with v. Otherwise, rotate the root with u.

We now bound the number of rotations in the second phase. Let $L(T)$ be the *left path* of a BST T, i.e., the maximal sequence $v_1, v_2, \ldots$ where v_1 is the root and v_{i+1} is the left child of v_i. Let the *right path* $R(T)$ be defined similarly. Observe that each step in the second phase increases the quantity $2|L(T)| + |R(T)|$ by one. Since $2|L(T)| + |R(T)| \leq 2n + 1$ for every BST T, the total number of rotations is linear.

The rotations only involve the root, its children, and its grandchildren, so (i) holds. To show (ii), suppose a node v never becomes the root. At the start of the second phase, v is in the right subtree of the root, and at the end, v is in the left subtree of the root. Thus, v must pass from the right subtree to the left subtree of the root at some point. The only way this can happen without v becoming the root is that v is in the left subtree of the right child of the root, and we rotate at the root. But in the second phase we only rotate at the root when the right child of the root has no left child, a contradiction. Thus, (ii) holds. ◀

▶ **Lemma 4.3.** *Let T be an arbitrary search tree on G. Then T can be transformed into some $A(S, \pi)$ using $2m + \mathcal{O}(n)$ rotations.*

Proof. An STG has the form $A(S, \pi)$ if and only if each leg node has no spine ancestor. The basic idea of the proof is to apply rotations between spine nodes to eventually bring each spine node to the root (of the spine BST). At any point, if the current root of the spine BST has a leg node as a child, rotate it with the leg node. We refer to all such rotations between two spine node rotations as the `cleanup` step. By Observation 3.2, every leg node is transported to the top this way.

The problem with this approach is that we can only rotate light edges in the spine BST, and the only edges that we know must be light are the edges between the BST root and its children. However, if we extend our `cleanup` step to consider leg nodes that are somewhat deeper in the tree (that is, nodes with two spine ancestors instead of just one), we guarantee that all BST edges *near* the BST root are light. This allows us to apply Lemma 4.2 to bring each spine node to the root. We now describe the sequence of rotations more formally, starting with the `cleanup` step.

Let T' be the current STG. Let `cleanup`(T') be the following sequence of rotations: As long as there is a leg node ℓ with a spine parent p and at most two spine ancestors, rotate (p, ℓ). Arbitrarily resolve conflicts. Let T'' be the STG after applying `cleanup`. Clearly, no spine node s with $\mathrm{depth}_{\mathrm{bst}(T'')}(s) \leq 2$ has a leg node child in T'', so all edges in $\mathrm{bst}(T'')$ involving the root or its children are light. Moreover, each leg node that is touched by `cleanup` is rotated at most twice, and afterwards has no spine node ancestors.

Let $\mathcal{X}$ be the sequence of rotations obtained by applying Lemma 4.2 to $\mathrm{bst}(T)$. We first apply `cleanup` to T, then apply the spine rotations in $\mathcal{X}$, with a `cleanup` step after each spine rotation. Since each rotation in $\mathcal{X}$ involves either the root of the spine BST or one of its children, the rotation is applied to a light edge. Thus, the whole sequence can indeed be applied to T.

The number of rotations is at most $2m + \mathcal{O}(n)$. Indeed, since no rotation between spine nodes can change the parent of a leg node (see Observation 3.2), each leg node is only touched in a single `cleanup` step (where it is rotated above all spine nodes), and only twice in that `cleanup` step. The length of $\mathcal{X}$ is $\mathcal{O}(n)$ by Lemma 4.2.

Finally, we show that the final tree is indeed of the form $A(S, \pi)$. For this, it suffices to show that each leg node that has at least one spine ancestor in T is touched in some `cleanup` step. Suppose that such a leg node ℓ is not involved in a `cleanup` step. Without loss of generality, let the parent p of ℓ be a spine node. By Observation 3.2, since ℓ is not touched in a `cleanup` step and we never rotate between leg nodes, p stays the parent of ℓ throughout the sequence of rotations. However, then p will be the root of $\mathrm{bst}(T)$ at some point, so ℓ is rotated upwards by the next `cleanup` step, a contradiction. ◀

It remains to show how to transform $A(S, \pi)$ into an STG without free leg nodes.

▶ **Lemma 4.4.** *Let $T = A(S, \pi)$. Then there is a sequence of $\mathcal{O}(n + H'(G) \cdot m)$ rotations that transform T into a search tree without free leg nodes.*

Proof. Since every edge in $\mathrm{bst}(T)$ is light, we can first transform $\mathrm{bst}(T)$ into an arbitrary BST S' using $\mathcal{O}(n)$ rotations. We will later specify S'. Let $T' = A(S', \pi)$ be the resulting STG. Now pick the lowest leg node $\ell_{i,j}$ in T', and rotate it down until it is bound (i.e., a child of s_i). Clearly, this requires $\mathrm{depth}_{\mathrm{bst}(T')}(s_i) = \mathrm{depth}_{S'}(s_i)$ rotations. Repeat this until all leg nodes are bound.

The total number of rotations is

$$\mathcal{O}(n) + \sum_{i=0}^{n} m_i \cdot \mathrm{depth}_{S'}(s_i).$$

This is precisely $\mathrm{cost}(S', m_1, m_2, \ldots, m_n)$, the cost of *accessing* each $i \in [n]$ with frequency m_i in the static BST S'. As such, if we choose S' to be the optimal static BST for these frequencies, we need $\mathcal{O}(n) + \text{OPT-ST}(m_1, m_2, \ldots, m_n) \leq \mathcal{O}(n) + 2H'(G) \cdot m$ rotations, by Lemma 2.1. ◀

Lemmas 4.1, 4.3, and 4.4 together imply the upper bound in Theorem 1.1.

In the proof of Lemma 4.4, we essentially treat the leg nodes as queries to our optimal static BST, where a leg node $\ell_{i,j}$ queries the spine node s_i. Rotating the leg nodes down is akin to moving down the pointer in the static BST model. Here, the pointer always points at the parent of the one leg node that has a spine node parent.

Observe that we can similarly implement the dynamic BST model as rotations transforming $A(S, \pi)$ into a search tree without bound leg nodes, simply by allowing spine node rotations (BST rotations) in between leg node rotations (pointer moves). If the dynamic BST algorithm wants to rotate the single heavy edge in the spine BST of our STG, we have to move the leg node out of the way (and back afterwards), but this only adds a constant-factor overhead. Thus we obtain a generalization of the dynamic BST model, where we can start processing queries before finishing previous ones (although the way "pointers" work in this model is not very intuitive).

Let $\sigma = \sigma(\pi)$ be the sequence of spine nodes obtained by replacing every leg node in π by its adjacent spine node. Our observations imply that transforming $A(S, \pi)$ into an STG without free leg nodes requires no more than $\mathcal{O}(\mathrm{OPT}(S, \sigma))$ rotations, and Lemma 4.4 essentially uses the fact that $\mathrm{OPT}(S, \sigma) \leq \text{OPT-ST}(m_1, m_2, \ldots, m_n)$. In the next section, we show that *Wilber's first lower bound* [22] for $\mathrm{OPT}(S, \sigma)$ also holds for our generalized model.

5 Lower bound

We start by defining a variant of Wilber's first lower bound and proving that it is equal to the Shannon entropy of the query distribution in the worst case (up to a constant factor). Then, we show that it also bounds the rotation distance between $A(S, \sigma)$ and $B(S)$ if σ is the worst-case ordering and S is a suitable search tree.

5.1 Wilber's first lower bound for binary search trees

Let S be a binary search tree on n nodes, let $\sigma = (x_1, x_2, \ldots, x_m)$ be a sequence of queries, and let u be a node of S. Then we define $\lambda(S, u, \sigma)$ as follows. If u has at most one child, then $\lambda(S, u, \sigma) = 0$. Otherwise, let v, w be the children of u and write $A = V(S_u) = \{u\} \cup V(S_v) \cup V(S_w)$. Let $\sigma|_A$ be the sequence obtained from σ by removing all elements not in A. Now $\lambda(S, u, \sigma)$ is defined as the number of times the sequence $\sigma|_A$ switches between an element of $V(S_v)$, an element of $V(S_w)$, and u. More formally, $\lambda(S, u, \sigma)$ is the number of pairs of adjacent values x, y in σ such that neither $x, y \in V(S_v)$, nor $x, y \in V(S_w)$, nor $x = y = u$. Let $\Lambda(S, \sigma) = \sum_{u \in V(S)} \lambda(S, u, \sigma)$.

For convenience, define $\lambda'(S, u, \sigma)$ as $\lambda(S, u, \sigma)$ plus the number of occurrences of u in σ, and let $\Lambda'(S, \sigma) = \sum_{u \in V(S)} \lambda'(S, u, \sigma) = \Lambda(S, \sigma) + m$. It is known that $\mathrm{OPT}(S, \sigma) \in \Omega(\Lambda'(S, \sigma))$. This is not tight in general [7, 13]. Still, Wilber [22] showed that if σ is the *bit reversal permutation*, then $\Lambda'(S, \sigma) \in \Theta(n \log n)$ for all S. This bound is already matched by a balanced static tree, so, on that sequence, Wilber's bound is tight and dynamic BSTs do not perform better than balanced trees. We now generalize this result to arbitrary distributions.

▶ **Lemma 5.1.** *Let $n \in \mathbb{N}_+$, let $m_1, m_2, \ldots, m_n \in \mathbb{N}_0$, and let $m = \sum_{i=1}^{n} m_i$. Then there is a BST S on $[n]$ and a sequence σ of length m where each $i \in [n]$ occurs precisely m_i times, such that $\Lambda'(S, \sigma) \geq \frac{1}{2} H(m_1, m_2, \ldots, m_n) \cdot m$.*

Proof. We recursively construct a BST $S_{p,q}$ on each interval $[p, q]$ with $1 \leq p \leq q \leq n$, and in the end set $S = S_{1,n}$. The construction is essentially the approximately optimal static BST construction due to Mehlhorn [16].

Fix p and q. Let $k = q - p + 1$ be the number of nodes in $S_{p,q}$, and, for each i with $p \leq i \leq q$, let $a_i = \sum_{j=p}^{i-1} m_j$ and $b_i = \sum_{j=i+1}^{q} m_j$. We claim that there exists an $i \in [p, q]$ such that $m_i + \min(a_i, b_i) \geq \frac{k}{2}$. Suppose not. Then, for each i, we have either (1) $m_i + a_i < \frac{k}{2} < b_i$ or (2) $m_i + b_i < \frac{k}{2} < a_i$. Observe that for $i = p$, (2) cannot hold, and likewise for $i = q$, (1) cannot hold. Let i' be the maximum index where (1) holds. Then, $i < q$ and $m_i + a_i < b_i = m_{i+1} + b_{i+1} < a_{i+1} = m_i + a_i$, a contradiction.

Choose $i \in [p, q]$ such that $m_i + \min(a_i, b_i) \geq \frac{k}{2}$. Make i the root of $S_{p,q}$, and attach the recursively constructed subtrees $S_{p,i-1}$ and $S_{i+1,q}$ as the left and right child to it (for $p' > q'$, we let $S_{p',q'}$ be the empty BST).

Let $c(p, q) = \sum_{j=p}^{q} m_j \cdot \mathrm{depth}_{S_{p,q}}(j)$. Intuitively, $c(p, q)$ is the cost of accessing the relevant nodes within $S_{p,q}$. We now recursively construct a sequence $\sigma_{p,q}$ such that $c(p, q) \leq 2\Lambda'(S_{p,q}, \sigma_{p,q})$.

First, if $p = q$, then let $\sigma_{p,q}$ simply consist of m_p times the element p. Clearly, $c(p,q) = m_p = \Lambda'(S_{p,q}, \sigma_{p,q})$. Otherwise, let i be the root of $S_{p,q}$. We construct $\sigma_{p,q}$ by combining $\sigma_{p,i-1}$, $\sigma_{i+1,q}$, and the m_i occurrences of the element i as follows. Start with the m_i occurrences of i, then alternate between $\sigma_{p,i-1}$ and $\sigma_{i+1,q}$ for as long as possible, and finally append the remaining elements from either $\sigma_{p,i-1}$ or $\sigma_{i+1,q}$. Since $\sigma_{p,i-1}$ has length a_i, and $\sigma_{i+1,q}$ has length b_i, we have $\lambda'(S_{p,q}, i, \sigma_{p,q}) \geq m_i + \min(a_i, b_i) \geq \frac{k}{2}$.

For each $j \in [p, i-1]$, we have $\mathrm{depth}_{S_{p,q}}(j) = 1 + \mathrm{depth}_{S_{p,i-1}}(j)$, and similarly for each $j \in [i+1, q]$, we have $\mathrm{depth}_{S_{p,q}}(j) = 1 + \mathrm{depth}_{S_{i+1,q}}(j)$. Thus, by induction,

$$\begin{aligned} c(p,q) &= m_i + a_i + c(p, i-1) + b_i + c(i+1, q) \\ &\leq k + 2\Lambda'(S_{p,i-1}, \sigma_{p,i-1}) + 2\Lambda'(S_{i+1,q}, \sigma_{i+1,q}) \\ &\leq 2\lambda'(S_{p,q}, i, \sigma_{p,q}) + 2\Lambda'(S_{p,i-1}, \sigma_{p,i-1}) + 2\Lambda'(S_{i+1,q}, \sigma_{i+1,q}) \leq 2\Lambda'(S_{p,q}, \sigma_{p,q}). \end{aligned}$$

Now let $S = S_{1,n}$ and $\sigma = \sigma_{1,n}$. We have $c(1,n) \leq 2\Lambda'(S, \sigma)$, and by Lemma 2.1, we know that $c(1,n) \geq \mathrm{cost}(m_1, m_2, \ldots, m_n) \geq H(m_1, m_2, \ldots, m_n) \cdot m$. This concludes the proof. ◀

5.2 Wilber's lower bound for rotation distance

We now show that the rotation distance between $A(S, \pi)$ and $B(S)$ is at least $\frac{1}{2}\Lambda'(S, \sigma(\pi))$, where $\sigma(\pi)$ is defined as in Section 4, by replacing the leaf nodes in π with their adjacent spine nodes. Our proof is based on the lower bound on the diameter of tree associahedra by Cardinal, Langerman, and Pérez-Lantero [3, Lemma 8].

▶ **Lemma 5.2.** *Let $G = C(m_1, m_2, \ldots, m_n)$ be a caterpillar, let S be a BST on the spine nodes of G, and let π be an ordering of the leg nodes of G. Then, transforming $A(S, \pi)$ into $B(S)$ requires at least $\frac{1}{2}\Lambda'(S, \sigma(\pi))$ rotations.*

Proof. Write $T = A(S, \pi)$, $T' = B(S)$, $\sigma = \sigma(\pi)$, and let r be the root of S. Let the set D consist of r and its adjacent legs, i.e., if $r = s_i$, then $D = \{s_i\} \cup \{\ell_{i,j} \mid j \in [m_i]\}$. Suppose r has two children u and v. Then $G \setminus D$ has two connected components, one consisting of the spine nodes $V(S_u)$ and all adjacent legs, and the other consisting of $V(S_v)$ and all adjacent legs. Call the former E and latter F. If r has only one child u, let E consist of $V(S_u)$ and all adjacent legs, and let $F = \emptyset$. If r has no children, let $E = F = \emptyset$. Note that D, E, F form a partition of $V(G)$.

We first consider the rotations within each of the three sets. By Lemma 2.2, we can simply sum up the number of rotations required to transform $T[D]$, $T[E]$, and $T[F]$ into $T'[D]$, $T'[E]$, $T'[F]$, respectively.

$T[D]$ consists of the spine node r and m_r free leg nodes, and $T'[D]$ consists of r and m_r bound leg nodes. Thus, we need m_r rotations to make all leg nodes bound.

If $E \neq \emptyset$, observe that $T[E] = A(S_u, \pi|_E)$ and $T'[E] = B(S_u)$, so we need $\frac{1}{2}\Lambda'(S_u, \sigma|_E) = \frac{1}{2}\Lambda'(S_u, \sigma)$ rotations by induction. If $F \neq \emptyset$, we similarly get a lower bound of $\frac{1}{2}\Lambda'(S_v, \sigma)$ to transform $T[F]$ into $T'[F]$.

We now show that there are at least $\frac{1}{2}\lambda(S, r, \sigma)$ rotations between *different* parts of the partition D, E, F. If $\lambda(S, r, \sigma) = 0$, this is trivially true, so suppose $\lambda(S, r, \sigma) > 0$ and thus $E \neq \emptyset$.

Define the *alternation number* of a path P in a search tree on G as the number of edges (x, y) in P such that x and y are in different parts of the partition D, E, F. Define the alternation number $\mathrm{alt}(T^*)$ of a search tree T^* on G as the maximum alternation number among all paths starting at the root in T^*. Observe that $\mathrm{alt}(T') = 1$, and that $\mathrm{alt}(T) \geq \lambda(S, r, \sigma) + 1$, since the leg nodes in T have $\lambda(S, r, \sigma)$ alternations by definition, and there is one more alternation from r to $E \neq \emptyset$.

We now show how rotations affect the alternation number. Consider a rotation between the nodes x and y, and a node z. The path from the root to z before and after the rotation may only differ if it contains x or y (or both), and only in one of the following ways:

- x is inserted before y, or y is inserted before x.
- x is deleted before y, or y is deleted before x.
- x and y are swapped (and are neighbors).

It is easy to see that if $x, y \in D$, or $x, y \in E$, or $x, y \in F$, then the rotation (x, y) does not affect the alternation number, and otherwise, it can only differ by at most two. This means that we need at least $\frac{1}{2}|\mathrm{alt}(T) - \mathrm{alt}(T')| \geq \frac{1}{2}\lambda(S, r, \sigma)$ rotations not within one of the sets D, E, or F. The total number of rotations is thus at least (setting $\Lambda'(S', \sigma) = 0$ if S' is empty):

$$m_r + \frac{1}{2}\Lambda'(S_u, \sigma) + \frac{1}{2}\Lambda'(S_v, \sigma) + \frac{1}{2}\lambda(S, r, \sigma) \geq \frac{1}{2}\Lambda'(S, \sigma). \qquad \blacktriangleleft$$

Lemmas 5.1 and 5.2 together imply that $\delta(\mathcal{A}(G)) \geq \frac{1}{4}H(G) \cdot m$. As mentioned in the introduction, Manneville and Pilaud [15] proved that $\delta(\mathcal{A}(G)) \in \Omega(m + n)$. This concludes the proof of the lower bound.

6 Conclusion

In this paper, we determined the diameter of each caterpillar associahedron up to a constant, revealing a surprising connection to searching in static and dynamic binary search trees. In particular, transforming $A(S, \pi)$ into $B(S)$ via rotations can be seen as a generalization of serving the access sequence $\sigma(\pi)$ in a dynamic BST. The number of rotations required is between Wilber's first lower bound $\Lambda(S, \sigma(\pi))$ and $\mathrm{OPT}(S, \sigma(\pi))$, begging the question whether other lower bounds for OPT hold in our generalized model, or whether it perhaps matches Λ or OPT. Results in this direction could give new insight into the dynamic BST model.

References

1. Benjamin Aram Berendsohn and László Kozma. Splay trees on trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1875–1900, 2022. doi:10.1137/1.9781611977073.75.
2. Prosenjit Bose, Jean Cardinal, John Iacono, Grigorios Koumoutsos, and Stefan Langerman. Competitive online search trees on trees. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1878–1891, 2020. doi:10.1137/1.9781611975994.115.
3. Jean Cardinal, Stefan Langerman, and Pablo Pérez-Lantero. On the diameter of tree associahedra. *The Electronic Journal of Combinatorics*, 2018. doi:10.37236/7762.
4. Jean Cardinal, Lionel Pournin, and Mario Valencia-Pabon. Bounds on the diameter of graph associahedra. *Procedia Computer Science*, 195:239–247, 2021. Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium. doi:10.1016/j.procs.2021.11.030.
5. Jean Cardinal, Lionel Pournin, and Mario Valencia-Pabon. Diameter estimates for graph associahedra. *arXiv e-prints*, 2021. arXiv:2106.16130.
6. Michael Carr and Satyan L. Devadoss. Coxeter complexes and graph-associahedra. *Topology and its Applications*, 153:2155–2168, 2004. doi:10.1016/j.topol.2005.08.010.
7. Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the Strong Wilber 1 Bound for Binary Search Trees. In Jarosław Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.33.

8 S. Cleary. Restricted rotation distance between binary trees. *Inf. Process. Lett.*, 84:333–338, 2002. doi:10.1016/S0020-0190(02)00315-0.

9 Erik D. Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505, 2009. doi:10.1137/1.9781611973068.55.

10 Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality – almost. *SIAM Journal on Computing*, 37(1):240–251, 2007. doi:10.1137/S0097539705447347.

11 D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, March 1971. doi:10.1007/BF00264289.

12 Jussi Kujala and Tapio Elomaa. The cost of offline binary search tree algorithms and the complexity of the request sequence. *Theoretical Computer Science*, 393(1):231–239, 2008. doi:10.1016/j.tcs.2007.12.015.

13 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber's bounds for dynamic optimality. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.68.

14 J.M. Lucas. Canonical forms for competitive binary search tree algorithms. Technical report DCS-TR-250, Department of Computer Science, Hill Center for the Mathematical Sciences, Busch Campus, Rutgers University, New Brunswick, New Jersey 08903, 1988.

15 Thibault Manneville and Vincent Pilaud. Graph properties of graph associahedra. *Séminaire Lotharingien de Combinatoire*, 73, 2015. URL: https://www.mat.univie.ac.at/~slc/wpapers/s73mannpil.html.

16 Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, December 1975. doi:10.1007/BF00264563.

17 J. Ian Munro. On the competitiveness of linear search. In Mike S. Paterson, editor, *Algorithms – ESA 2000*, pages 338–345, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/3-540-45253-2_31.

18 L. Pournin. The asymptotic diameter of cyclohedra. *Israel Journal of Mathematics*, 219:609–635, 2014. doi:10.1007/s11856-017-1492-0.

19 Lionel Pournin. The diameter of associahedra. *Advances in Mathematics*, 259:13–42, 2014. doi:10.1016/j.aim.2014.02.035.

20 D. Sleator and R. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32:652–686, 1985. doi:10.1145/3828.3835.

21 Daniel D Sleator, Robert E Tarjan, and William P Thurston. Rotation distance, triangulations, and hyperbolic geometry. *Journal of the American Mathematical Society*, 1(3):647–681, 1988. doi:10.1090/S0894-0347-1988-0928904-4.

22 Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM journal on Computing*, 18(1):56–67, 1989. doi:10.1137/0218004.