Fast Approximation of Search Trees on Trees with Centroid Trees

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

Ishay Golinsky
Blavatnik School of Computer Science, Tel Aviv University, Israel

Haim Kaplan
Blavatnik School of Computer Science, Tel Aviv University, Israel

László Kozma
Institut für Informatik, Freie Universität Berlin, Germany

Abstract

Search trees on trees (STTs) generalize the fundamental binary search tree (BST) data structure: in STTs the underlying search space is an arbitrary tree, whereas in BSTs it is *a path*. An optimal BST of size n can be computed for a given distribution of queries in $\mathcal{O}(n^2)$ time [Knuth, Acta Inf. 1971] and *centroid* BSTs provide a nearly-optimal alternative, computable in $\mathcal{O}(n)$ time [Mehlhorn, SICOMP 1977].

By contrast, optimal STTs are not known to be computable in polynomial time, and the fastest constant-approximation algorithm runs in $\mathcal{O}(n^3)$ time [Berendsohn, Kozma, SODA 2022]. Centroid trees can be defined for STTs analogously to BSTs, and they have been used in a wide range of algorithmic applications. In the unweighted case (i.e., for a uniform distribution of queries), the centroid tree can be computed in $\mathcal{O}(n)$ time [Brodal, Fagerberg, Pedersen, Östlin, ICALP 2001; Della Giustina, Prezza, Venturini, SPIRE 2019]. These algorithms, however, do not readily extend to the weighted case. Moreover, no approximation guarantees were previously known for centroid trees in either the unweighted or weighted cases.

In this paper we revisit centroid trees in a general, weighted setting, and we settle both the algorithmic complexity of constructing them, and the quality of their approximation. For constructing a weighted centroid tree, we give an *output-sensitive* $\mathcal{O}(n \log h) \subseteq \mathcal{O}(n \log n)$ time algorithm, where h is the height of the resulting centroid tree. If the weights are of polynomial complexity, the running time is $\mathcal{O}(n \log \log n)$. We show these bounds to be optimal, in a general decision tree model of computation. For approximation, we prove that the cost of a centroid tree is at most *twice* the optimum, and this guarantee is best possible, both in the weighted and unweighted cases. We also give tight, fine-grained bounds on the approximation-ratio for bounded-degree trees and on the approximation-ratio of more general α-centroid trees.

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

Keywords and phrases centroid tree, search trees on trees, approximation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.19

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `arxiv.org/abs/2209.08024` [6]

Funding *Benjamin Aram Berendsohn*: DFG grant KO 6140/1-1.
Ishay Golinsky: ISF grant no. 1595/19 and the Blavatnik Research Foundation.
Haim Kaplan: ISF grant no. 1595/19 and the Blavatnik Research Foundation.
László Kozma: DFG grant KO 6140/1-1.

1 Introduction

Search trees on trees (STTs) are a far-reaching generalization of binary search trees (BSTs), modeling the exploration of tree-shaped search spaces. Given an undirected tree $\mathcal{T}$, an STT on $\mathcal{T}$ is a tree rooted at an arbitrary vertex r of $\mathcal{T}$, with subtrees built recursively on the components resulting after removing r from $\mathcal{T}$, see Figure 1 for an example. BSTs correspond to the special case where the underlying tree $\mathcal{T}$ is a *path*.

STTs and, more generally, search trees on graphs arise in several different contexts and have been studied under different names: *tubings* [14], *vertex rankings* [20, 8, 23], *ordered colorings* [35], *elimination trees* [43, 49, 2, 9]. STTs have been crucial in many algorithmic applications, e.g., in pattern matching and counting [24, 37, 27], cache-oblivious data structures [4, 25], tree clustering [26], geometric visibility [30], planar point location [29], distance oracles [16]. They arise in matrix factorization (e.g., see [22, § 12]), and have also been related to the competitive ratio in certain online hitting set problems [23].

Similarly to the setting of BSTs, a natural goal is to find an STT in which the expected depth of a vertex is as small as possible; we refer to such a tree as an *optimal tree*, noting that it is not necessarily unique. This optimization task can be studied both for the uniform probability distribution over the vertices, and for the more general case of an arbitrary distribution given as input. We refer to the first as the *unweighted* and the second as the *weighted* problem.

For BSTs, both the unweighted and the weighted problems are well-understood. In the unweighted case, a simple balanced binary tree achieves the optimum. In the weighted case, an optimal tree on n vertices can be found in time $\mathcal{O}(n^2)$ by Knuth's algorithm [36], a textbook example of dynamic programming. No faster algorithm is known in general, although Larmore's algorithm [40] achieves better bounds under certain regularity assumptions on the weights; for example, if the probability assigned to each vertex is $\Omega(1/n)$, then the optimum can be found in time $\mathcal{O}(n^{1.591})$.

By contrast, the complexity of computing an optimal STT is far less understood. Even in the unweighted case, no polynomial-time algorithm is known, and the problem is not known to be NP-hard even with arbitrary weights. Recently, a PTAS was given for the weighted problem [7], but its running time for obtaining a $(1+\varepsilon)$-approximation of the optimal STT is $\mathcal{O}(n^{1+2/\varepsilon})$, which is prohibitive for reasonably small values of ε. Note that the apparently easier problem of minimizing the *maximum depth* of a vertex, i.e., computing the *treedepth* of a tree, can be solved in linear time by Schäffer's algorithm [50], and treedepth itself has many algorithmic applications, e.g., see [47, § 6,7].

Centroid trees. Given the relatively high cost of computing optimal binary search trees, research has turned already half a century ago to efficient approximations. Mehlhorn has shown [44, 45] that a simple BST that can be computed in $\mathcal{O}(n)$ time closely approximates the optimum. More precisely, both the optimum cost and the cost of the obtained tree are in $[H/\log(3), H+1]$, where H is the binary entropy of the input distribution.[1] Alternatively, the cost can be upper bounded by $\mathtt{OPT} + \log(\mathtt{OPT}) + \log e$, where $\mathtt{OPT}$ is the cost of the optimal tree. Observe that this means that the approximation ratio gets arbitrarily close to 1 as $\mathtt{OPT}$ goes to infinity.[2]

[1] All logarithms in this paper are base 2.

[2] Results for BSTs are sometimes presented in a more general form, where the input distribution also accounts for *unsuccessful searches*, i.e., it may assign non-zero probabilities to the *gaps* between neighboring vertices and outside the two extremes. Extending such a model to STTs is straightforward, but perhaps less natural in the case of general trees, we therefore omit it for the sake of simplicity, and consider only successful searches.

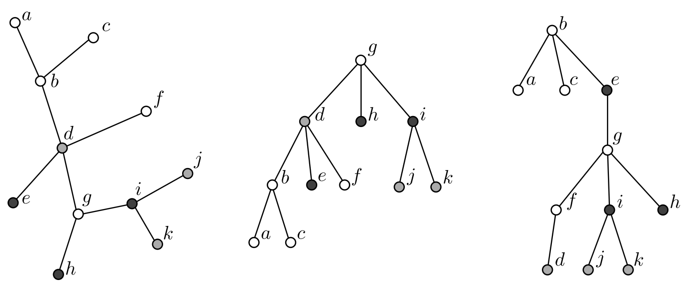


Figure 1 (*Left.*) Tree $\mathcal{T}$. (*Middle.*) Centroid tree of $\mathcal{T}$. (*Right.*) A different STT on $\mathcal{T}$. Colors indicate weights (probabilities), $w(e) = w(h) = w(i) = 0.15$, $w(d) = w(j) = w(k) = 0.10$, and all other vertices have weight 0.05. Observe that the centroid tree is (in this example) unique.

The BST that achieves the above guarantees is built by recursively picking roots such as to make the weights of the left and right subtrees "as equal as possible". This is a special case of a *centroid tree*, defined as follows. Given a tree $\mathcal{T}$, a *centroid* of $\mathcal{T}$ is a vertex whose removal from $\mathcal{T}$ results in components with weight *at most half* of the total weight of $\mathcal{T}$. A centroid tree is built by iteratively finding a centroid and recursing on the components resulting after its removal. See Figure 1 for an example.

The fact that an (unweighted) centroid always exists was already shown in the 19-th century by C. Jordan [34]. We sketch the easy, constructive argument that also shows the existence of a weighted centroid: start at an arbitrary vertex of $\mathcal{T}$ and, as long as the current vertex is not a centroid, move one edge in the direction of the component with largest weight. It is not hard to see that the procedure succeeds, visiting each vertex at most once.

A straightforward implementation of the above procedure finds an unweighted centroid tree in $\mathcal{O}(n \log n)$ time. This running time has been improved to $\mathcal{O}(n)$ by carefully using data structures [11, 28]. The run-time guarantees however, do not readily generalize from the unweighted to the weighted setting. Intuitively, the difficulty lies in the fact that in the weighted case, the removal of a centroid vertex may split the tree in a very unbalanced way, leaving up to $n - 1$ vertices in one component. Thus, a naive recursive approach will take $\Theta(n^2)$ time in the worst case.

Most algorithmic applications of STTs, including those mentioned before, rely on centroid trees. It is therefore surprising that nothing appears to be known about how well the centroid tree approximates the optimal STT in either the unweighted or weighted cases. In this paper we prove that the centroid tree is a 2-approximation of the optimal STT, and that the factor 2 is, in general, best possible, both in the unweighted and weighted settings. As our main result, we also show a more precise bound on the approximation ratio of centroid trees, in terms of the maximum degree of the underlying tree $\mathcal{T}$.[3]

Before stating our results, we need a few definitions. Consider an undirected, unrooted tree $\mathcal{T}$ given as input, together with a *weight function* $w : V(\mathcal{T}) \to \mathbb{R}_{\geq 0}$. For convenience, for any subgraph $\mathcal{H}$ of $\mathcal{T}$, we denote $w(\mathcal{H}) = \sum_{x \in V(\mathcal{H})} w(x)$. (To interpret the weights as probabilities, we need the condition $w(\mathcal{T}) = 1$. It is, however, often convenient to relax this requirement and allow arbitrary non-negative weights, which is the approach we will take.)

[3] In their recent paper on dynamic STTs, Bose, Cardinal, Iacono, Koumoutsos, and Langerman [10] remark that the ratio between the costs of the centroid- and optimal trees may be unbounded. In light of our results, this observation is erroneous. It is true, however, that a centroid tree built using the uniform distribution may be far from the optimum w.r.t. a different distribution.

A *search tree* on $\mathcal{T}$ is a rooted tree T with vertex set $V(\mathcal{T})$ whose root is an arbitrary vertex $r \in V(\mathcal{T})$. The children of r in T are the roots of search trees built on the connected components of the forest $\mathcal{T} - r$. A tree consisting of a single vertex admits only itself as a search tree. It follows from the definition that for all x, the subtree T_x of T rooted at x induces a connected subgraph $\mathcal{T}[V(T_x)]$ of $\mathcal{T}$, and moreover, T_x is a search tree on $\mathcal{T}[V(T_x)]$.

The *cost* of a search tree T on $\mathcal{T}$ is $\texttt{cost}_w(T) = \sum_{x \in V(T)} w(x) \cdot \texttt{depth}_T(x)$, where the depth of the root is taken to be 1. The *optimum cost* $\texttt{OPT}(\mathcal{T}, w)$ is the minimum of $\texttt{cost}_w(T)$ over all search trees T of $\mathcal{T}$.

A vertex $v \in V(\mathcal{T})$ is a *centroid* if for all components $\mathcal{H}$ of $\mathcal{T} - v$, we have $w(\mathcal{H}) \le w(\mathcal{T})/2$. A search tree T of $\mathcal{T}$ is a *centroid tree* if vertex x is a centroid of $\mathcal{T}[V(T_x)]$ for all $x \in V(\mathcal{T})$. In general, the centroid tree is not unique, and centroid trees of the same tree can have different costs.[4] We denote by $\texttt{cent}(\mathcal{T}, w)$ the *maximum* cost of a centroid tree of $(\mathcal{T}, w)$, with weight function w.

We can now state our approximation guarantee for centroid trees.

▶ **Theorem 1.** *Let $\mathcal{T}$ be a tree, $w : V(\mathcal{T}) \to \mathbb{R}_{\ge 0}$, and $m = w(\mathcal{T})$. Then*

$$\texttt{cent}(\mathcal{T}, w) \le 2 \cdot \texttt{OPT}(\mathcal{T}, w) - m.$$

We show that this result is optimal, including in the additive term. Moreover, the constant factor 2 cannot be improved even for unweighted instances.

▶ **Theorem 2.**

(i) *For every $\varepsilon > 0$ there is a sequence of instances $(\mathcal{T}_n, w_n)$ with $w_n(\mathcal{T}_n) = 1$, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\texttt{cost}_{w_n}(C_n) \ge 2 \cdot \texttt{OPT}(\mathcal{T}_n, w_n) - 1 - \varepsilon.$$

(ii) *There is a sequence of instances $(\mathcal{T}_n, w_n)$, where w_n is the uniform distribution on $V(\mathcal{T}_n)$, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\lim_{n \to \infty} \frac{\texttt{cost}_{w_n}(C_n)}{\texttt{OPT}(\mathcal{T}_n, w_n)} = 2.$$

In both cases $\lim\limits_{n \to \infty} \texttt{OPT}(\mathcal{T}_n, w_n) = \infty$.

Note that the fact that $\lim_{n \to \infty} \texttt{OPT}(\mathcal{T}_n, w_n) = \infty$ in Theorem 2 establishes that the *asymptotic* approximation ratio is 2. By this we mean that every bound of the form $\texttt{cent} \le c \cdot \texttt{OPT} + o(\texttt{OPT})$ must have $c \ge 2$.

We next show a stronger guarantee when the underlying tree has bounded degree.

▶ **Theorem 3.** *Let $\mathcal{T}$ be a tree, $w : V(\mathcal{T}) \to \mathbb{R}_{\ge 0}$, and let Δ be the* maximum degree *of $\mathcal{T}$. Then*

$$\texttt{cent}(\mathcal{T}, w) \le \left(2 - \frac{1}{2^{\Delta}}\right) \cdot \texttt{OPT}(\mathcal{T}, w).$$

We complement this result by two lower bounds. The first establishes the tightness of the approximation ratio. The second shows a (slightly smaller) lower bound on the approximation ratio for instances where OPT is unbounded.

[4] Consider, for instance the two different centroid trees of a path on four vertices, with weights $(0.2, 0.3, 0.2, 0.3)$.

▶ **Theorem 4.** *Let $\Delta \geq 3$ be integer.*

(i) *There is a sequence of instances $(\mathcal{T}_n, w_n)$ such that $\mathcal{T}_n$ has maximum degree at most Δ, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\lim_{n\to\infty} \frac{\mathtt{cost}_{w_n}(C_n)}{\mathtt{OPT}(\mathcal{T}_n, w_n)} = 2 - \frac{1}{2^\Delta}.$$

(ii) *There is a sequence of instances $(\mathcal{T}_n, w_n)$ such that $\mathcal{T}_n$ has maximum degree at most Δ, $\lim\limits_{n\to\infty} \mathtt{OPT}(\mathcal{T}_n, w_n) = \infty$, $w_n(\mathcal{T}_n) = 1$, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\mathtt{cost}_{w_n}(C_n) \geq \left(2 - \frac{4}{2^\Delta}\right) \cdot \mathtt{OPT}(\mathcal{T}_n, w_n) - 1.$$

We remark that Theorem 4(i) does not exclude the possiblity of a bound of the form $\mathtt{cent} \leq c \cdot \mathtt{OPT} + o(\mathtt{OPT})$, where $c < 2 - \frac{1}{2^\Delta}$, as here $\mathtt{OPT}(\mathcal{T}_n, w_n)$ is bounded. Part (ii), however, establishes that a bound of the form $\mathtt{cent} \leq c \cdot \mathtt{OPT} + o(\mathtt{OPT})$ must have $c \geq 2 - \frac{4}{2^\Delta}$. We leave open the problem of closing the gap in the asymptotic approximation ratio in terms of Δ.

Computing centroid trees. On the algorithmic side, we show that the weighted centroid tree can be computed in $\mathcal{O}(n \log n)$ time. Previously, the fastest known constant-approximation algorithm [7] took $\mathcal{O}(n^3)$ time (similarly achieving an approximation ratio of 2). The main step of our algorithm, finding the weighted centroid of a tree, is achievable in $\mathcal{O}(\log n)$ time, assuming that the underlying tree is stored in a *top tree* data structure [1]. Iterating this procedure in combination with known algorithms for *constructing* and *splitting* top trees yields the algorithm that runs in $\mathcal{O}(n \log n)$ time. As our main algorithmic result, we also develop an improved, *output-sensitive* algorithm, with running time $\mathcal{O}(n \log h)$, where h is the height of the resulting centroid tree, yielding a running time $\mathcal{O}(n \log\log n)$ in the typical case when the height is $\mathcal{O}(\log n)$.

▶ **Theorem 5.** *Let $\mathcal{T}$ be a tree on n vertices and w be a weight function. We can compute a centroid tree of $(\mathcal{T}, w)$ in time $\mathcal{O}(n \log h)$, where h is the height of the computed centroid tree.*

One may ask whether the weighted centroid tree can be computed in linear time, similarly to the unweighted centroid tree, or to the weighted centroid BST. We show that, assuming a general decision tree model of computation, this is not possible, and the algorithm of Theorem 5 is optimal for all n and h (up to a constant factor). Our lower bound on the running time applies, informally, to any deterministic algorithm in which the input weights affect program flow in the form of binary decisions, involving arbitrary computable functions.

More precisely, consider a tree $\mathcal{T}$ on n vertices. We say that a *binary decision tree $D_\mathcal{T}$ solves* $\mathcal{T}$ for a class of weight functions $\mathcal{W}$ mapping $V(\mathcal{T})$ to $\mathbb{R}_{\geq 0}$, if the leaves of $D_\mathcal{T}$ are search trees on $\mathcal{T}$, every branching of $D_\mathcal{T}$ is of the form "$f(w)>0$?" for some computable function $f : \mathcal{W} \to \{-1, +1\}$, and for every weight function $w \in \mathcal{W}$, starting from the root of $D_\mathcal{T}$ and following branchings down the tree, we reach a leaf T of $D_\mathcal{T}$ that is a valid centroid tree for $(\mathcal{T}, w)$. The height of $D_\mathcal{T}$ is then a lower bound on the worst-case running time.

▶ **Theorem 6.** *Let $h \geq 3$ and $n \geq h + 1$ be integers. Then there is a tree $\mathcal{T}$ on at most n vertices and a class $\mathcal{W}$ of weight functions on $V(\mathcal{T})$ such that for every $w \in \mathcal{W}$, every centroid tree of $(\mathcal{T}, w)$ has height h, and every binary decision tree that solves $\mathcal{T}$ for $\mathcal{W}$ has height $\Omega(n \log h)$.*

We can nonetheless improve the running time, when the weights are restricted in certain (natural) ways. We define the *spread* σ of a weight function w as the ratio between the total weight $w(\mathcal{T})$, and the smallest *non-zero* weight of a vertex. As we show, $\mathcal{O}(n \log h) \subseteq \mathcal{O}(n \log\log(\sigma + n))$ and therefore, when $\sigma \in n^{\mathcal{O}(1)}$ (for instance, if the weights are integers stored in RAM words), we obtain a running time of $\mathcal{O}(n \log\log n)$.

When many vertices have zero weight, we obtain further improvements, e.g., if only $\mathcal{O}(n/\log n)$ of the weights are non-zero, we can compute a centroid tree in $\mathcal{O}(n)$ time, even if the height h is large. The precise statement of these refined bounds and the discussion of their optimality are available in the full version of this paper [6].

Approximate centroid trees. Finally, we consider the approximation guarantees of a generalized form of centroid trees. Let us call a vertex v of a tree $\mathcal{T}$ an α-centroid, for $0 \le \alpha \le 1$, if $w(\mathcal{H}) \le \alpha \cdot w(\mathcal{T})$, for all components $\mathcal{H}$ of $\mathcal{T} - v$. An α-centroid tree is an STT in which every vertex x is an α-centroid of its subtree $\mathcal{T}[V(T_x)]$.

Observe that the standard centroid tree is a $\frac{1}{2}$-centroid tree, and all STTs are 1-centroid trees. Also note that an α-centroid is a β-centroid for all $\beta \ge \alpha$ and that the existence of an α-centroid is not guaranteed for $\alpha < \frac{1}{2}$ (consider a single edge with the two endpoints having the same weight). On the other hand, an α-centroid for $\alpha < \frac{1}{2}$, if it exists, is unique, and therefore the α-centroid tree is also unique. To see this, consider an α-centroid c that splits $\mathcal{T}$ into components $\mathcal{T}_1, \dots, \mathcal{T}_k$. If an alternative α-centroid c' were in component $\mathcal{T}_i$, then its removal would yield a component containing all vertices in $\mathcal{T} - V(\mathcal{T}_i)$, of weight at least $(1-\alpha) \cdot w(\mathcal{T}) > \alpha \cdot w(\mathcal{T})$.

Denote by $\texttt{cent}^{\alpha}(\mathcal{T}, w)$ the maximum cost of an α-centroid tree of $(\mathcal{T}, w)$, or 0 if no α-centroid tree exists. We refine our guarantee from Theorem 1 to approximate centroid trees:

▶ **Theorem 7.** *Let $\mathcal{T}$ be a tree, $w : V(\mathcal{T}) \to \mathbb{R}_{\ge 0}$, $m = w(\mathcal{T})$. We have*

$$(i)\quad \texttt{cent}^{\alpha}(\mathcal{T}, w) \le \frac{1}{1-\alpha} \cdot \texttt{OPT}(\mathcal{T}, w) - \frac{\alpha}{1-\alpha} m, \qquad \text{for } \alpha \in (0,1),$$

$$(ii)\quad \texttt{cent}^{\alpha}(\mathcal{T}, w) \le \frac{1}{2-3\alpha} \cdot \texttt{OPT}(\mathcal{T}, w) - \frac{3\alpha-1}{2-3\alpha} m, \qquad \text{for } \alpha \in \left[\frac{1}{3}, \frac{1}{2}\right].$$

Note that the second bound is a strengthening of the first when $\alpha < \frac{1}{2}$. In particular, for $\alpha \le \frac{1}{3}$, it implies that an α-centroid tree is *optimal*, if it exists.

We show that the result is tight when $\alpha \ge \frac{1}{2}$ by proving a matching lower bound.

▶ **Theorem 8.** *For every $\alpha \in [\frac{1}{2}, 1)$ there is a sequence of instances $(\mathcal{T}_n, w_n)$ with $\lim_{n\to\infty} \texttt{OPT}(\mathcal{T}_n, w_n) = \infty$, $w_n(\mathcal{T}_n) = 1$ and*

$$\texttt{cent}^{\alpha}(\mathcal{T}_n, w_n) \ge \frac{1}{1-\alpha} \cdot \texttt{OPT}(\mathcal{T}_n, w_n) - \frac{\alpha}{1-\alpha}.$$

Note that if $\alpha > \frac{1}{2}$, we cannot prove such a lower bound for *all* α-centroid trees of $(\mathcal{T}_n, w_n)$ (as in Theorem 2), since a $\frac{1}{2}$-centroid tree exists and has stronger approximation guarantees according to Theorem 1.

Finally, we argue that every optimal STT is a $\frac{2}{3}$-centroid tree. A special case of this result (for BSTs) was shown by Hirschberg, Larmore, and Molodowitch [32], who also showed that the ratio $\frac{2}{3}$ is tight (in the special case of BSTs, and thus, also for STTs).

▶ **Theorem 9.** *Let T be an optimal STT of $(\mathcal{T}, w)$. Then, T is a $\frac{2}{3}$-centroid tree of $(\mathcal{T}, w)$.*

Structure of the paper. In this extended abstract we omit some of the proofs, discussions, and technical details which are included in the full paper [6]. In Section 2 we state a number of results needed in the proofs. The general upper and lower bounds on the approximation ratio of centroid trees (Theorems 1 and 2) are proved in Section 3. These results can be seen as a warm-up towards the fine-grained bounds on the approximation ratio of centroid trees (Theorems 3 and 4), which we prove in Section 4. The algorithmic results (Theorem 5) are discussed in Section 5, with the details of the output-sensitive algorithm, the lower bounds (Theorem 6), and further extensions available in the full paper. Results on α-centroids (Theorems 7, 8, and 9) are proved in the full paper. In Section 6 we conclude with open questions.

Related work. Different models of searching in trees have also been considered, e.g., the one where we query edges instead of vertices [3, 39, 46, 48], with connections to searching in posets [42, 31]. In the edge-query setting, Cicalese, Jacobs, Laber and Molinaro [17, 18] study the problem of minimizing the average search time of a vertex, and show this to be an NP-hard problem [17]. They also show that an "edge-centroid" tree (in their terminology, a greedy algorithm) gives a 1.62-approximation of the optimum [18].

STTs generalize BSTs, therefore it is natural to ask to what extent the theory developed for BSTs can be extended to STTs. Defining a natural *rotation* operation on STTs, Bose, Cardinal, Iacono, Koumoutsos, and Langerman [10] develop an $O(\log\log n)$ competitive dynamic STT, analogously to Tango BSTs [19]. In a similar spirit, Berendsohn and Kozma [7] generalize Splay trees [51] to STTs. The rotation operation on STTs naturally leads to the definition of *tree associahedra*, a combinatorial structure that extends the classical associahedron defined over BSTs or other Catalan-structures. Properties of tree- and more general graph associahedra have been studied in [14, 21, 15, 12, 13, 5].

Searching in trees and graphs has also been motivated with applications, including file system synchronisation [3, 46], software testing [3, 46], asymmetric communication protocols [38], VLSI layout [41], and assembly planning [33].

2 Preliminaries

Given a graph G, we denote by $V(G)$ its set of vertices, by $E(G)$ its set of edges, and by $\mathbb{C}(G)$ its set of connected components. If $v \in V(G)$, denote by $N_G(v)$ the set of neighbors of v in G, and $\deg_G(v) = |N_G(v)|$. For $S \subseteq V(G)$, denote by $G[S]$ the subgraph of G induced by S, and for brevity, $G - v = G[V(G) - \{v\}]$, and $G - S = G[V(G) - S]$.

The following observation is straightforward.

▶ **Observation 10.** *Let T be a search tree on $\mathcal{T}$, $w : V(\mathcal{T}) \to \mathbb{R}_{\geq 0}$, $m = w(\mathcal{T})$ and $r = \texttt{root}(T)$. For each component $\mathcal{H} \in \mathbb{C}(\mathcal{T} - r)$, denote by $T_{\mathcal{H}}$ the subtree of T rooted at the unique child of r in $\mathcal{H}$. Then*

$$\texttt{cost}_w(T) = m + \sum_{\mathcal{H}\in\mathbb{C}(\mathcal{T}-r)} \texttt{cost}_w(T_{\mathcal{H}}) \geq m + \sum_{\mathcal{H}\in\mathbb{C}(\mathcal{T}-r)} \texttt{OPT}(\mathcal{H}, w).$$

Projection of a search tree. For a rooted tree T and a vertex $v \in V(T)$, we denote by $\texttt{Path}_T(v)$ the set of vertices on the path in T from $\texttt{root}(T)$ to v, including both endpoints. Our upper bounds require the following notion of *projection* of a search tree.

▶ **Theorem 11.** *Let T be a search tree on $\mathcal{T}$ and $\mathcal{H}$ a connected subgraph of $\mathcal{T}$. There is a unique search tree $T|_{\mathcal{H}}$ on $\mathcal{H}$ such that for every $v \in V(\mathcal{H})$,*

$$\texttt{Path}_{T|_{\mathcal{H}}}(v) = \texttt{Path}_T(v) \cap V(\mathcal{H}).$$

▶ **Definition 12** (Projection)**.** *Let T be a search tree on $\mathcal{T}$ and $\mathcal{H}$ a connected subgraph of $\mathcal{T}$. We call $T|_{\mathcal{H}}$, whose existence is established by Theorem 11, the projection of T to $\mathcal{H}$.*

Tie-breaking. Our lower bounds require the following tie-breaking procedure.

▶ **Lemma 13.** *Let $\mathcal{T}$ be a tree, $w : V(\mathcal{T}) \to \mathbb{R}_{\geq 0}$ a weight function and let C be a centroid tree of $(\mathcal{T}, w)$. For every $\varepsilon > 0$ there exists a weight function $w' : V(\mathcal{T}) \to \mathbb{R}_{\geq 0}$ such that C is the unique centroid tree of $(\mathcal{T}, w')$ and $\|w' - w\|_\infty < \varepsilon$.*

Centroid and median. A certain concept of a *median vertex* of a tree has been used previously in the literature. If $\mathcal{T}$ is a tree with positive vertex weights and positive edge weights, then the median of $\mathcal{T}$ is the vertex v minimizing the quantity $\sum_{u \neq v} w(u) \cdot d(u, v)$. Here $w(u)$ is the weight of the vertex u, and $d(u, v)$ is the distance from u to v, i.e., the sum of the edge weights on the path from u to v. We show that if all edge-weights are 1, then medians are precisely centroids.

▶ **Lemma 14.** *Let $\mathcal{T}$ be a graph and w be a weight function on $V(\mathcal{T})$. For each $u \in V(\mathcal{T})$, define $W(u) = \sum_{v \in V(\mathcal{T})} d_{\mathcal{T}}(u, v) \cdot w(v)$, where $d_{\mathcal{T}}(u, v)$ denotes the number of edges on the path from u to v in $\mathcal{T}$. Then $c \in V(\mathcal{T})$ is a centroid of $(\mathcal{T}, w)$ if and only if $W(c)$ is minimal.*

Proofs to Theorem 11 and Lemmas 13 and 14 are available in the full paper [6].

3 Approximation guarantees for general trees

In this section we prove the general upper bound and lower bounds the approximation quality of centroid trees. We start with the upper bound.

▶ **Theorem 1.** *Let $\mathcal{T}$ be a tree, $w : V(\mathcal{T}) \to \mathbb{R}_{\geq 0}$, and $m = w(\mathcal{T})$. Then*

$$\texttt{cent}(\mathcal{T}, w) \leq 2 \cdot \texttt{OPT}(\mathcal{T}, w) - m.$$

We prove the following lemma.

▶ **Lemma 15.** *Let c be a centroid of $(\mathcal{T}, w)$ and $m = w(\mathcal{T})$. Then*

$$\texttt{OPT}(\mathcal{T}, w) \geq \frac{m}{2} + \frac{w(c)}{2} + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T} - c)} \texttt{OPT}(\mathcal{H}, w). \tag{1}$$

Proof. Let T be an arbitrary search tree on $\mathcal{T}$. We will show that $\texttt{cost}_w(T)$ is at least the right hand side of Equation (1).

Denote $r = \texttt{root}(T)$. If $r = c$, using Observation 10, we have

$$\texttt{cost}_w(T) \geq m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T} - c)} \texttt{OPT}(\mathcal{H}, w),$$

which implies the claim. Assume therefore that $r \neq c$. Denote by $\mathcal{H}^*$ the connected component of $\mathcal{T} - c$ where r is. The contribution of vertices of $\mathcal{H}^*$ to $\texttt{cost}_w(T)$ is at least $\texttt{cost}_w(T|_{\mathcal{H}^*})$. For $\mathcal{H} \in \mathbb{C}(\mathcal{T} - c)$, $\mathcal{H} \neq \mathcal{H}^*$ and $v \in V(\mathcal{H})$, we have $\texttt{Path}_T(v) \supseteq \{r\} \cup \texttt{Path}_{T|_{\mathcal{H}}}(v)$, therefore the contribution of vertices of every $\mathcal{H} \neq \mathcal{H}^*$ is at least $w(\mathcal{H}) + \texttt{cost}_w(T|_{\mathcal{H}})$. Finally, the contribution of c is at least $2w(c)$, since both $c, r \in \texttt{Path}_T(c)$. Summing the contributions of all the vertices, we get

$$\begin{aligned}\texttt{cost}_w(T) &\geq 2w(c) + \texttt{cost}_w(T|_{\mathcal{H}^*}) + \sum_{\mathcal{H} \neq \mathcal{H}^*} (w(\mathcal{H}) + \texttt{cost}_w(T|_{\mathcal{H}})) \\ &\geq m - w(\mathcal{H}^*) + w(c) + \sum_{\mathcal{H}} \texttt{OPT}(\mathcal{H}, w) \\ &\geq \frac{m}{2} + w(c) + \sum_{\mathcal{H}} \texttt{OPT}(\mathcal{H}, w),\end{aligned}$$

where the last inequality follows from c being a centroid. ◀

Proof of Theorem 1. The proof is by induction on the number of vertices. When $|V(\mathcal{T})| = 1$ we have

$$2 \cdot \texttt{OPT}(\mathcal{T}, w) - m = 2m - m = m = \texttt{cent}(\mathcal{T}, w), \quad \text{as required.}$$

Assume $|V(\mathcal{T})| > 1$. Let C be a centroid tree on $\mathcal{T}$ and $c = \texttt{root}(C)$. Using Observation 10 and the induction hypothesis we have:

$$\begin{aligned}\texttt{cost}_w(C) &\leq m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T}-c)} \texttt{cent}(\mathcal{H}, w) \\ &\leq m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T}-c)} (2 \cdot \texttt{OPT}(\mathcal{H}, w) - w(\mathcal{H})) \\ &= w(c) + 2 \cdot \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T}-c)} \texttt{OPT}(\mathcal{H}, w),\end{aligned}$$

therefore it is enough to show that

$$w(c) + 2 \cdot \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T}-c)} \texttt{OPT}(\mathcal{H}, w) \leq 2 \cdot \texttt{OPT}(\mathcal{T}, w) - m,$$

which is just a re-arrangement of Lemma 15. This concludes the proof. ◀

Next, we prove the lower bounds on the approximation quality of centroid trees, showing the tightness of Theorem 1. We note that in the *edge-query model* of search trees a 2-approximation was shown in [18] using techniques similar to those in the proof of Theorem 1. In contrast to that result, however, our approximation guarantee is best possible.

▶ **Theorem 2.**

(i) *For every $\varepsilon > 0$ there is a sequence of instances $(\mathcal{T}_n, w_n)$ with $w_n(\mathcal{T}_n) = 1$, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\texttt{cost}_{w_n}(C_n) \geq 2 \cdot \texttt{OPT}(\mathcal{T}_n, w_n) - 1 - \varepsilon.$$

(ii) *There is a sequence of instances $(\mathcal{T}_n, w_n)$, where w_n is the uniform distribution on $V(\mathcal{T}_n)$, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\lim_{n\to\infty} \frac{\texttt{cost}_{w_n}(C_n)}{\texttt{OPT}(\mathcal{T}_n, w_n)} = 2.$$

In both cases $\lim_{n\to\infty} \texttt{OPT}(\mathcal{T}_n, w_n) = \infty$.

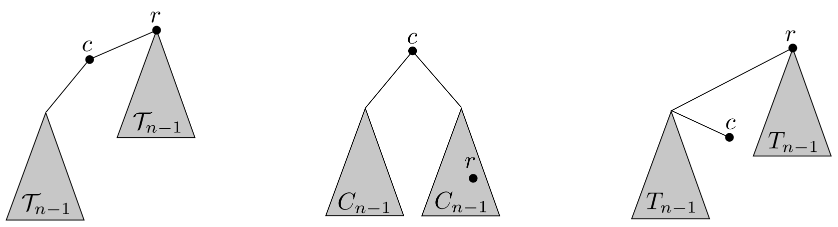


Figure 2 Illustration of the proof of Theorem 2(i). Vertex r is the root of $\mathcal{T}_n$. (*Left.*) The underlying tree $\mathcal{T}_n$. (*Middle.*) The entroid tree C_n of $\mathcal{T}_n$. (*Right.*) The search tree T_n.

As the proofs of both parts of Theorem 2 use the same construction with only slightly different analyses, we prove here only part (i). The proof of part (ii) appears in the full paper [6].

We proceed by constructing a sequence $(\mathcal{T}_n, w_n)$ such that for *some* centroid tree C_n,

$$\mathtt{cost}_{w_n}(C_n) \geq 2 \cdot \mathtt{OPT}(\mathcal{T}_n, w_n) - 1.$$

Using Lemma 13, we can then add an arbitrarily small perturbation to w_n to make C_n the *unique* centroid tree. (Observe that for every search tree T, $\mathtt{cost}_w(T)$ is continuous in w, therefore so is $\mathtt{OPT}(\mathcal{T}, w)$.)

The sequence $(\mathcal{T}_n, w_n)$ is constructed recursively as follows. For the sake of the construction we view $\mathcal{T}_n$ as a rooted tree. The base case $\mathcal{T}_0$ is a tree with a single vertex v and $w_0(v) = 1$. For $n > 0$, take two copies $(\mathcal{A}, w_{\mathcal{A}})$ and $(\mathcal{B}, w_{\mathcal{B}})$ of $(\mathcal{T}_{n-1}, w_{n-1})$. Connect the roots of $\mathcal{A}$ and $\mathcal{B}$ to a new vertex c. Finally, set $\mathtt{root}(\mathcal{T}_n) = \mathtt{root}(\mathcal{A})$ (see Figure 2). We define w_n as follows. (observe that $w_n(\mathcal{T}_n) = 1$, by induction on n.)

$$w_n(v) = \begin{cases} 0, & v = c \\ \frac{1}{2} w_{\mathcal{A}}(v), & v \in V(\mathcal{A}) \\ \frac{1}{2} w_{\mathcal{B}}(v), & v \in V(\mathcal{B}). \end{cases}$$

Let C_n denote the search tree on $\mathcal{T}_n$ obtained by setting c as the root and recursing. Observe that C_n is a centroid tree of $(\mathcal{T}_n, w_n)$.

▶ **Lemma 16.** *The following hold*

(a) $\mathtt{cost}_{w_n}(C_n) = n + 1$,

(b) $\lim_{n\to\infty} \mathtt{OPT}(\mathcal{T}_n, w_n) = \infty$.

Proof. Let $c_n = \mathtt{cost}_{w_n}(C_n)$. Clearly, $c_0 = 1$. Assume $n > 0$. Let $C_{\mathcal{A}}$ and $C_{\mathcal{B}}$ be search trees on $\mathcal{A}$ and $\mathcal{B}$ respectively, each a copy of C_{n-1}. By construction of C_n we have

$$c_n = 1 + \frac{1}{2}\mathtt{cost}_{w_{\mathcal{A}}}(C_{\mathcal{A}}) + \frac{1}{2}\mathtt{cost}_{w_{\mathcal{B}}}(C_{\mathcal{B}}) = 1 + c_{n-1},$$

and (a) follows by induction.

Using (a) and Theorem 1, part (b) follows:

$$\mathtt{OPT}(\mathcal{T}_n, w_n) \geq \frac{c_n + 1}{2} = \frac{n}{2} + 1 \to \infty. \qquad ◀$$

Next, in order to bound $\mathtt{OPT}(\mathcal{T}_n, w_n)$ from above, we construct a sequence of search trees T_n on $\mathcal{T}_n$. For $n = 0$, tree T_0 is a single vertex. Assume $n > 0$. Let $\mathcal{A}$, $\mathcal{B}$, and c be as in the definition of $\mathcal{T}_n$. Let $T_{\mathcal{A}}$ and $T_{\mathcal{B}}$ be search trees over $\mathcal{A}$ and $\mathcal{B}$ respectively, each a copy of T_{n-1}. Denote $r_{\mathcal{A}} = \mathtt{root}(\mathcal{A})$ and $r_{\mathcal{B}} = \mathtt{root}(\mathcal{B})$. Tree T_n is obtained by adding an edge from $r_{\mathcal{A}}$ to $r_{\mathcal{B}}$ and an edge from $r_{\mathcal{B}}$ to c, and setting $\mathtt{root}(T_n) = r_{\mathcal{A}}$.

▶ **Lemma 17.** $\texttt{cost}_{w_n}(T_n) = \frac{n}{2} + 1$.

Proof. Denote $t_n = \texttt{cost}_{w_n}(T_n)$. Clearly $t_0 = 1$. Assume $n > 0$. The contribution of vertices of $\mathcal{A}$ to t_n is exactly $\frac{1}{2}\texttt{cost}_{w_{\mathcal{A}}}(T_{\mathcal{A}}) = \frac{t_{n-1}}{2}$. Since r is an ancestor of all vertices in $\mathcal{B}$, the contribution of these vertices to t_n is exactly $\frac{1}{2}(1 + \texttt{cost}_{w_{\mathcal{B}}}(T_{\mathcal{B}})) = \frac{1+t_{n-1}}{2}$. Summing the contribution of all vertices, we get $t_n = t_{n-1} + \frac{1}{2}$ and the claim follows. ◀

Proof of Theorem 2(i). By Lemma 17, $\texttt{OPT}(\mathcal{T}_n, w_n) \leq \frac{n}{2} + 1$. Together with Lemma 16, the claim follows. ◀

4 Approximation guarantees for trees with bounded degrees

In this section we show the upper and lower bounds on the approximation quality of centroid trees when the underlying tree $\mathcal{T}$ has bounded degree. We start with the upper bound.

▶ **Theorem 3.** *Let $\mathcal{T}$ be a tree, $w : V(\mathcal{T}) \to \mathbb{R}_{\geq 0}$, and let Δ be the* maximum degree *of $\mathcal{T}$. Then*

$$\texttt{cent}(\mathcal{T}, w) \leq \left(2 - \frac{1}{2^{\Delta}}\right) \cdot \texttt{OPT}(\mathcal{T}, w).$$

For simplicity, in what follows we omit the weight function w from notations.

▶ **Lemma 18.** *Let C be a centroid tree of $\mathcal{T}$ such that $\texttt{cost}(C) = \texttt{cent}(\mathcal{T})$. Let $P = (v_0, v_1, \ldots, v_p)$ be any path in C. Then*

$$\texttt{cent}(\mathcal{T}) \leq \left(2 - \frac{1}{2^p}\right) m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T} - P)} \texttt{cent}(\mathcal{H}).$$

Proof. By induction on p. For $p = 0$ we have

$$\texttt{cent}(\mathcal{T}) = m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T} - v_0)} \texttt{cent}(\mathcal{H}), \quad \text{as required.}$$

Assume now $p > 0$. Denote by $\tilde{\mathcal{T}}$ the connected component of $\mathcal{T} - v_0$ where v_1 is. Denote $\tilde{P} = (v_1, \ldots, v_p)$, and $\tilde{m} = w(\tilde{\mathcal{T}})$. Observe that $\tilde{m} \leq m/2$ and that $\mathbb{C}(\mathcal{T} - P) = \mathbb{C}(\tilde{\mathcal{T}} - \tilde{P}) \uplus (\mathbb{C}(\mathcal{T} - v_0) - \{\tilde{\mathcal{T}}\})$. By the induction hypothesis we have

$$\texttt{cent}(\tilde{\mathcal{T}}) \leq \left(2 - \frac{1}{2^{p-1}}\right) \tilde{m} + \sum_{\mathcal{H} \in \mathbb{C}(\tilde{\mathcal{T}} - \tilde{P})} \texttt{cent}(\mathcal{H}), \quad \text{therefore}$$

$$\begin{aligned}
\texttt{cent}(\mathcal{T}) &= m + \texttt{cent}(\tilde{\mathcal{T}}) + \sum_{\substack{\mathcal{H} \in \mathbb{C}(\mathcal{T} - v_0) \\ \mathcal{H} \neq \tilde{\mathcal{T}}}} \texttt{cent}(\mathcal{H}) \\
&\leq m + \left(2 - \frac{1}{2^{p-1}}\right) \tilde{m} + \sum_{\mathcal{H} \in \mathbb{C}(\tilde{\mathcal{T}} - \tilde{P})} \texttt{cent}(\mathcal{H}) + \sum_{\substack{\mathcal{H} \in \mathbb{C}(\mathcal{T} - v_0) \\ H \neq \tilde{\mathcal{T}}}} \texttt{cent}(\mathcal{H}) \\
&\leq m + \left(2 - \frac{1}{2^{p-1}}\right) \frac{m}{2} + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T} - P)} \texttt{cent}(\mathcal{H}) \\
&= \left(2 - \frac{1}{2^p}\right) m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T} - P)} \texttt{cent}(\mathcal{H}), \quad \text{as required.}
\end{aligned}$$

◀

Proof of Theorem 3. The proof is by induction on $|V(\mathcal{T})|$. Let T be any search tree on $\mathcal{T}$. We will show that $\texttt{cent}(\mathcal{T}) \leq \left(2 - \frac{1}{2^{\Delta}}\right) \texttt{cost}(T)$.

Denote $r = \texttt{root}(T)$. Let C be a centroid tree on $\mathcal{T}$ with $\texttt{cost}(C) = \texttt{cent}(\mathcal{T})$. Denote by $v_0, v_1, \ldots, v_d = r$ the vertices along the path to r in C. Denote $\mathcal{T}_i = \mathcal{T}[V(C_{v_i})]$. Observe that $r \in V(\mathcal{T}_d) \subseteq \cdots \subseteq V(\mathcal{T}_0) = V(\mathcal{T})$. For $i < d$, denote by $\mathcal{K}_i$ the connected component of $\mathcal{T} - r$ where v_i is. Denote by s_i the unique child of r in T such that $s_i \in V(\mathcal{K}_i)$, i.e., $V(T_{s_i}) = V(\mathcal{K}_i)$. Finally, denote by p the minimal i for which one of the following holds:

1. $v_i = r$, i.e., $i = d$,
2. $s_i \in V(\mathcal{T}_{i+1})$, or
3. there exists $j < i$ such that $\mathcal{K}_j = \mathcal{K}_i$, i.e., $s_j = s_i$.

Note that from the third condition above it follows that $p \leq \Delta$. Denote $P = (v_0, \ldots, v_p)$. We will prove the following.

▷ Claim 19.

$$\texttt{cost}(T) \geq m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T}-P)} \texttt{cost}(T|_{\mathcal{H}}).$$

Assume for now that Claim 19 holds. Using Lemma 18, the fact that $p \leq \Delta$, the induction hypothesis and Claim 19, we have

$$\begin{aligned}\texttt{cent}(\mathcal{T}) &\leq \left(2 - \frac{1}{2^p}\right) m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T}-P)} \texttt{cent}(\mathcal{H}) \\ &\leq \left(2 - \frac{1}{2^{\Delta}}\right) m + \sum_{\mathcal{H} \in \mathbb{C}(\mathcal{T}-P)} \left(2 - \frac{1}{2^{\Delta}}\right) \texttt{cost}(T|_{\mathcal{H}}) \\ &\leq \left(2 - \frac{1}{2^{\Delta}}\right) \texttt{cost}(T).\end{aligned}$$ ◀

Proof of Claim 19. The proof breaks into cases according to the defining condition of p.

Case 1. Assume $v_p = r$. For every $\mathcal{H} \in \mathbb{C}(\mathcal{T} - P)$ and $v \in V(\mathcal{H})$ we have $\texttt{Path}_T(v) \supseteq \{r\} \cup \texttt{Path}_{T|_{\mathcal{H}}}(v)$. The contribution of such v to $\texttt{cost}(T)$ is therefore at least $w(v)(1 + |\texttt{Path}_{T|_{\mathcal{H}}}(v)|)$. The contribution of each v_i to $\texttt{cost}(T)$ is at least $w(v_i)$. Summing the contribution of all vertices yields the required result. See Figure 3.

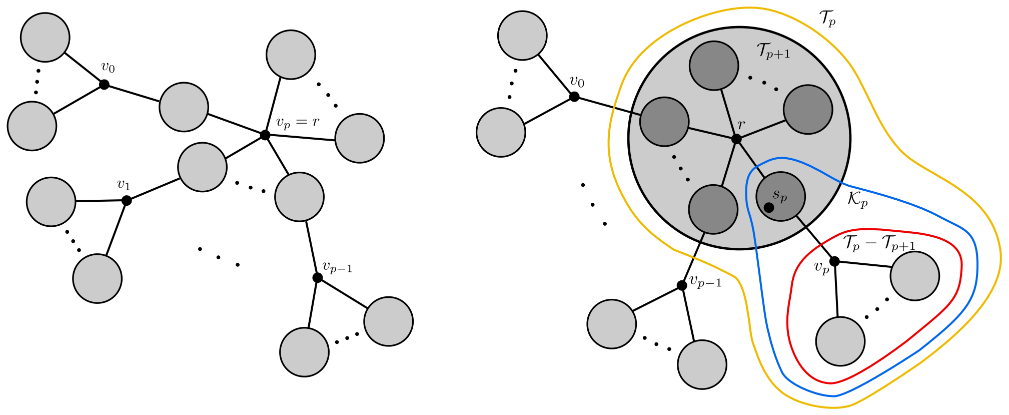
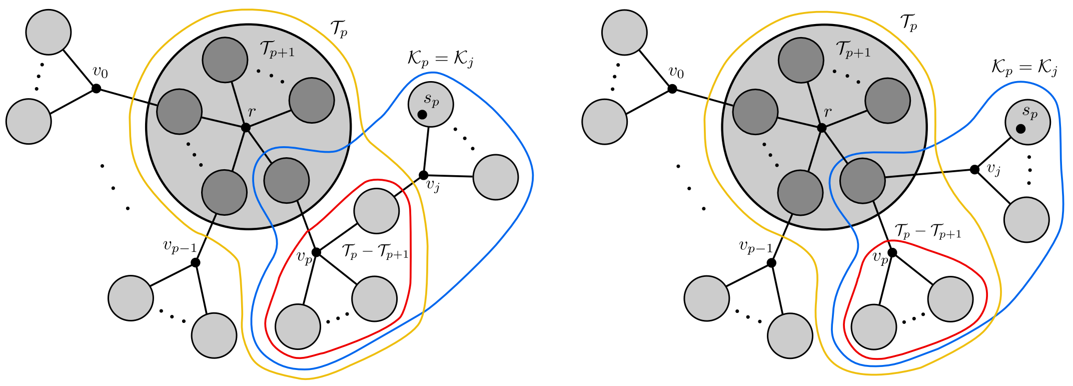


Figure 3 Illustration of the proof of Claim 19. Connected components of $\mathcal{T} - P$ are represented by light gray circles. (*Left.*) Case 1. (*Right.*) Case 2. Vertices in $\mathcal{T} - \mathcal{T}_p$ have r as ancestor. Vertices in $\mathcal{T}_p - \mathcal{T}_{p+1}$ have both r and s_p as ancestors.



$\mathcal{T}_p$
$\mathcal{T}_{p+1}$
$\mathcal{K}_p = \mathcal{K}_j$
v_0
r
s_p
v_j
v_{p-1}
v_p
$\mathcal{T}_p - \mathcal{T}_{p+1}$

Figure 4 Case 3 in the proof of Claim 19. (*Left.*) Sub-case where v_p is in the path in $\mathcal{T}$ between r and v_j. (*Right.*) Complementary sub-case. Connected components of $\mathcal{T} - P$ are represented by light gray circles. In both sub-cases, vertices in $\mathcal{T}_p - \mathcal{T}_{p+1}$ have both r and s_p as ancestors.

Case 2. Assume $s_p \in V(\mathcal{T}_{p+1})$. Denote by $\mathbb{C}_1$ the set of connected components of $\mathcal{T} - P$ that are not contained in $\mathcal{T}_p$. Denote $\mathbb{C}_2 = \mathbb{C}(\mathcal{T} - P) - \mathbb{C}_1$ (see Figure 3). For every $\mathcal{H} \in \mathbb{C}_1$, if $v \in V(\mathcal{H})$, then $\texttt{Path}_T(v) \supseteq \{r\} \cup \texttt{Path}_{T|_\mathcal{H}}(v)$. Therefore the contribution of vertices in $V(\mathcal{T} - \mathcal{T}_p)$ to $\texttt{cost}(T)$ is at least

$$m - w(\mathcal{T}_p) + \sum_{\mathcal{H} \in \mathbb{C}_1} \texttt{cost}(T|_\mathcal{H}). \tag{2}$$

For vertices $v \in V(\mathcal{T}_p - \mathcal{T}_{p+1})$ we have $\{r, s_p\} \subseteq \texttt{Path}_T(v)$. Therefore, using the fact that $w(\mathcal{T}_p - \mathcal{T}_{p+1}) \geq \frac{w(\mathcal{T}_p)}{2}$, the contribution of vertices in $V(\mathcal{T}_p)$ to $\texttt{cost}(T)$ is at least

$$2 \cdot w(\mathcal{T}_p - \mathcal{T}_{p+1}) + \sum_{\mathcal{H} \in \mathbb{C}_2} \texttt{cost}(T|_\mathcal{H}) \geq w(\mathcal{T}_p) + \sum_{\mathcal{H} \in \mathbb{C}_2} \texttt{cost}(T|_\mathcal{H}). \tag{3}$$

Summing Equation (2) and Equation (3) yields the required result.

Case 3. Assume that there is a $j < p$ such that $\mathcal{K}_p = \mathcal{K}_j$. Since p is minimal, we further assume that Case 2 did not occur for indices smaller that p. In particular, $s_p = s_j \notin V(\mathcal{T}_p)$. As in Case 2, the contribution of vertices in $\mathcal{T} - \mathcal{T}_p$ to $\texttt{cost}(T)$ is at least as in Equation (2). We have $r \notin V(\mathcal{T}_p - \mathcal{T}_{p+1})$ and $v_p \in V(\mathcal{T}_p - \mathcal{T}_{p+1}) \cap V(\mathcal{K}_p) \neq \emptyset$, therefore, since $\mathcal{T}_p - \mathcal{T}_{p+1}$ is connected, $V(\mathcal{T}_p - \mathcal{T}_{p+1}) \subseteq V(\mathcal{K}_p)$ (see Figure 4). It follows that vertices in $\mathcal{T}_p - \mathcal{T}_{p+1}$ have both r and s_p as ancestors. Since $w(\mathcal{T}_p - \mathcal{T}_{p+1}) \geq \frac{w(\mathcal{T}_p)}{2}$, the contribution of vertices in $\mathcal{T}_p$ is at least as in Equation (3). As in Case 2, the result follows by summing Equation (2) and Equation (3). $\lhd$

We now proceed to the lower bounds.

▶ **Theorem 4.** *Let $\Delta \geq 3$ be integer.*

(i) *There is a sequence of instances $(\mathcal{T}_n, w_n)$ such that $\mathcal{T}_n$ has maximum degree at most Δ, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\lim_{n \to \infty} \frac{\texttt{cost}_{w_n}(C_n)}{\texttt{OPT}(\mathcal{T}_n, w_n)} = 2 - \frac{1}{2^\Delta}.$$

(ii) *There is a sequence of instances $(\mathcal{T}_n, w_n)$ such that $\mathcal{T}_n$ has maximum degree at most Δ, $\lim_{n \to \infty} \texttt{OPT}(\mathcal{T}_n, w_n) = \infty$, $w_n(\mathcal{T}_n) = 1$, and for every centroid tree C_n of $(\mathcal{T}_n, w_n)$*

$$\texttt{cost}_{w_n}(C_n) \geq \left(2 - \frac{4}{2^\Delta}\right) \cdot \texttt{OPT}(\mathcal{T}_n, w_n) - 1.$$

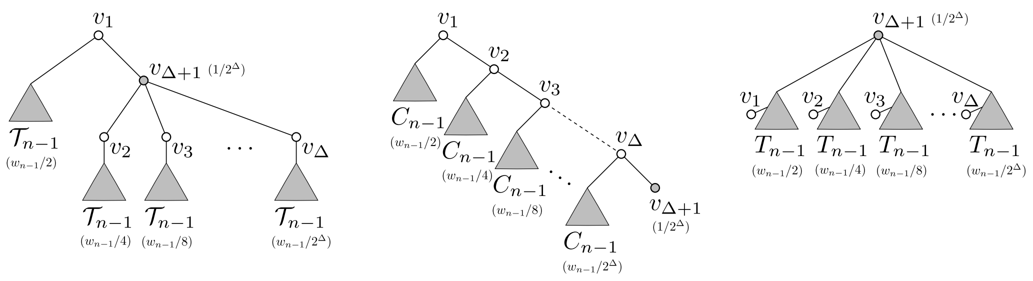

v_1
$v_{\Delta+1}$ $(1/2^\Delta)$
$\mathcal{T}_{n-1}$ $(w_{n-1}/2)$
v_2 v_3 $\cdots$ v_Δ
$\mathcal{T}_{n-1}$ $(w_{n-1}/4)$
$\mathcal{T}_{n-1}$ $(w_{n-1}/8)$
$\mathcal{T}_{n-1}$ $(w_{n-1}/2^\Delta)$
v_1 v_2 v_3 v_Δ
C_{n-1} $(w_{n-1}/2)$
C_{n-1} $(w_{n-1}/4)$
C_{n-1} $(w_{n-1}/8)$
C_{n-1} $(w_{n-1}/2^\Delta)$
$v_{\Delta+1}$ $(1/2^\Delta)$
$v_{\Delta+1}$ $(1/2^\Delta)$
v_1 v_2 v_3 $\cdots$ v_Δ
T_{n-1} $(w_{n-1}/2)$
T_{n-1} $(w_{n-1}/4)$
T_{n-1} $(w_{n-1}/8)$
T_{n-1} $(w_{n-1}/2^\Delta)$

Figure 5 Illustration of Theorem 4(i). (*Left.*) The underlying tree $\mathcal{T}_n$. (*Middle.*) The centroid tree C_n. (*Right.*) The search tree T_n.

Part (i). As in the proof of Theorem 2, it will suffice to prove Theorem 4(i) for *some* centroid tree C_n. Using Lemma 13, we can then add arbitrarily small perturbation to w_n, making C_n the unique centroid tree.

The sequence $(\mathcal{T}_n, w_n)$ of Theorem 4(i) is constructed recursively as follows. For the sake of the construction we regard $\mathcal{T}_n$ as a rooted tree. $\mathcal{T}_0$ is simply a single vertex v (which is the root) and $w_0(v) = 1$. For $n > 0$, $\mathcal{T}_n$ is constructed from Δ copies of $\mathcal{T}_{n-1}$ and $\Delta + 1$ additional vertices, $v_1, \dots, v_{\Delta+1}$, as shown in Figure 5 (left). The i'th copy of $\mathcal{T}_{n-1}$ gets the weight function $w_{n-1}/2^i$. We set $w_n(v_i) = 0$ for $1 \le i \le \Delta$ and $w_n(v_{\Delta+1}) = 1/2^\Delta$. Finally, we set $\text{root}(\mathcal{T}_n) = v_1$. By induction, $\mathcal{T}_n$ has maximal degree Δ and w_n is a distribution on $V(\mathcal{T}_n)$.

Let C_n be a search tree on $\mathcal{T}_n$ defined recursively as follows. Connect the vertices $v_1, \dots, v_{\Delta+1}$ to form a path and set v_1 as the root of C_n. Continue recursively on each connected component of $\mathcal{T} - \{v_1, \dots, v_{\Delta+1}\}$. (See Figure 5 (middle).) Observe that C_n is a centroid tree of $(\mathcal{T}_n, w_n)$.

▶ **Lemma 20.** *For all n,*

$$\texttt{cost}_{w_n}(C_n) = 2^{\Delta+1} - 1 - (2^{\Delta+1} - 2)\left(1 - \frac{1}{2^\Delta}\right)^n. \tag{4}$$

Proof. Denote $c_n = \texttt{cost}_{w_n}(C_n)$. Clearly $c_0 = 1$ as required. Let $n > 0$. For each i, the subtree of all the descendants of v_i in C_n has weight $1/2^{i-1}$. Therefore

$$c_n = \sum_{i=1}^{\Delta+1} \frac{1}{2^{i-1}} + \sum_{i=1}^{\Delta} \frac{1}{2^i} c_{n-1} = 2 - \frac{1}{2^\Delta} + \left(1 - \frac{1}{2^\Delta}\right) c_{n-1}.$$

It is straightforward to verify that the right hand side of Equation (4) is the solution to the recursive formula above. ◀

In order to upper bound $\texttt{OPT}(\mathcal{T}_n, w_n)$ we construct recursively a search tree T_n on $\mathcal{T}_n$. For $n > 0$, T_n is constructed by setting $v_{\Delta+1}$ as root and attaching to it Δ copies of T_{n-1}. The vertices $v_1, \dots, v_\Delta$ are finally attached as leaves of T_n, each at its unique valid place. See Figure 5 (right).

▶ **Lemma 21.** *For all n,*

$$\texttt{cost}_{w_n}(T_n) = 2^\Delta - 2^\Delta\left(1 - \frac{1}{2^\Delta}\right)^{n+1}. \tag{5}$$

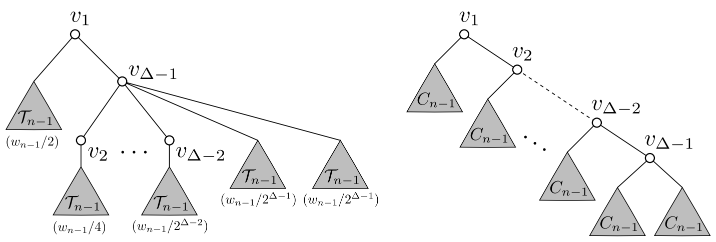


Figure 6 An illustration of Theorem 4(ii). (*Left.*) The underlying tree $\mathcal{T}_n$. (*Middle.*) The centroid tree C_n. (*Right.*) The search tree T_n.

Proof. Denote $t_n = \texttt{cost}_{w_n}(T_n)$. We have $t_0 = 1$. For $n > 0$, t_n obeys the recursive relation

$$t_n = 1 + \sum_{i=1}^{\Delta} \frac{1}{2^i} t_{n-1} = 1 + \left(1 - \frac{1}{2^\Delta}\right) t_{n-1},$$

of which the right hand side of Equation (5) is the solution. ◀

Proof of Theorem 4(i). Using Lemma 20 and Lemma 21,

$$\frac{\texttt{cost}_{w_n}(C_n)}{\texttt{OPT}(\mathcal{T}_n, w_n)} \geq \frac{\texttt{cost}_{w_n}(C_n)}{\texttt{cost}_{w_n}(T_n)} \to 2 - \frac{1}{2^\Delta}.$$ ◀

Observe that, as discussed in Section 1, $\texttt{OPT}(\mathcal{T}_n, w_n)/w_n(\mathcal{T}_n)$ is bounded.

Part (ii). To prove Theorem 4(ii), we repeat the recursive construction of Theorem 4(i) with a slight modification. As before, $\mathcal{T}_0$ is a tree with a single vertex. For $n > 0$, $(\mathcal{T}_n, w_n)$ is constructed from Δ weighted copies of $(\mathcal{T}_{n-1}, w_{n-1})$ and $\Delta - 1$ additional vertices, $v_1, \dots, v_{\Delta-1}$, each with weight 0, as shown in Figure 6 (left). We set $\texttt{root}(\mathcal{T}_n) = v_1$.

As before, the search tree C_n is defined by connecting the vertices $v_1, \dots, v_{\Delta-1}$ to a path, setting v_1 as root and recursing on the remaining connected component. Observe that C_n is a centroid tree of $(\mathcal{T}_n, w_n)$. (See Figure 6 (middle).) The search tree T_n is defined by setting $v_{\Delta-1}$ as root, attaching to it Δ copies of T_{n-1}, then adding the vertices $v_1, \dots, v_{\Delta-2}$ as leaves, each at its unique valid place. See Figure 6 (right).

▶ **Lemma 22.** *For all n,*
(a) $\texttt{cost}_{w_n}(C_n) = \left(2 - \frac{4}{2^\Delta}\right) \cdot n + 1$,
(b) $\texttt{cost}_{w_n}(T_n) = n + 1$.

The proof follows an analysis similar to that of Lemma 20 and Lemma 21.

Proof. Denote $c_n = \texttt{cost}_{w_n}(C_n)$ and $t_n = \texttt{cost}_{w_n}(T_n)$. Clearly $c_0 = t_0 = 1$. For $n > 0$ we have

$$c_n = \sum_{i=1}^{\Delta-1} \frac{1}{2^{i-1}} + \sum_{i=1}^{\Delta-2} \frac{1}{2^i} c_{n-1} + 2\frac{1}{2^{\Delta-1}} c_{n-1} = 2 - \frac{4}{2^\Delta} + c_{n-1}, \quad \text{and}$$

$$t_n = \sum_{i=1}^{\Delta-2} \frac{1}{2^i}(t_{n-1} + 1) + 2\frac{1}{2^{\Delta-1}}(t_{n-1} + 1) = 1 + t_{n-1},$$

and the lemma follows by induction. ◀

Proof of Theorem 4(ii). The fact that $\lim_{n\to\infty} \texttt{OPT}(\mathcal{T}_n, w_n) = \infty$ follows from Lemma 22 and Theorem 3 (or Theorem 1). Using Lemma 22 again, we have

$$\texttt{cost}_{w_n}(C_n) \geq \left(2 - \frac{4}{2^\Delta}\right) \texttt{OPT}(\mathcal{T}_n, w_n) - 1 + \frac{4}{2^\Delta}.$$

Using Lemma 13, for each n we can add small enough perturbation to w_n such that C_n is the unique centroid tree and the claimed bound holds. ◀

5 Computing centroid trees

In this section, we show how to compute centroid trees using the *top tree* framework of Alstrup, Holm, de Lichtenberg, and Thorup [1]. *Top trees* are a data structure used to maintain dynamic forests under insertion and deletion of edges. Most importantly, they expose a simple interface that allows the user to maintain information in the trees of the forest. For this, the user only needs to implement a small number of internal operations.

Alstrup et al. in particular show how to maintain the *median* of trees in $\mathcal{O}(\log n)$ per operation, see Section 2 for the definition of the median. As mentioned before, if all edge-weights are 1, then medians are precisely centroids (see Lemma 14).

▶ **Theorem 23** ([1, Theorem 3.6]). *We can maintain a forest with positive vertex weights on n vertices under the following operations:*
- *Add an edge between two given vertices u, v that are not in the same connected component;*
- *Remove an existing edge;*
- *Change the weight of a vertex;*
- *Retrieve a pointer to the tree containing a given vertex;*
- *Find the centroid of a given tree in the forest.*

Each operation requires $\mathcal{O}(\log n)$ time. A forest without edges and with n arbitrarily weighted vertices can be initialized in $\mathcal{O}(n)$ time.

Note that Theorem 23 only admits *positive* vertex weights, whereas we allowed zero-weight vertices. We show how to handle this problem in the full paper [6].

We now show how to use Theorem 23 to construct a centroid tree in $\mathcal{O}(n \log n)$ time.

▶ **Theorem 24.** *Given a tree $\mathcal{T}$ on n vertices and a positive weight function w, we can compute a centroid tree of $(\mathcal{T}, w)$ in $\mathcal{O}(n \log n)$ time.*

Proof. First build a top tree on $\mathcal{T}$ by adding the edges one-by-one, in $\mathcal{O}(n \log n)$ time. Then, find the centroid c, and remove each incident edge. Then, recurse on each newly created tree (except for the one containing only c). The algorithm finds each vertex precisely once and removes each edge precisely once, for a total running time of $\mathcal{O}(n \log n)$. ◀

Output-sensitive algorithm. We improve the algorithm given above to run in time $\mathcal{O}(n \log h)$, where n is the number of vertices in $\mathcal{T}$ and h is the height of the computed centroid tree.

The main idea of the algorithm is inspired by the linear-time algorithm for *unweighted* centroids by Della Giustina, Prezza, and Venturini [28], with a number of further technical challenges. Instead of building a top tree on the whole tree $\mathcal{T}$, we first split $\mathcal{T}$ into connected subgraphs of size roughly h, and build a top tree on each component. Contracting each component into a single vertex yields *super-vertices* in a *super-tree*. Each search for a centroid consists of a global search and a local search: We first find the super-vertex containing the centroid, then we find the centroid within that super-vertex. After finding the centroid, we remove it, which may split up the super-vertex into multiple super-vertices with a top tree each, and also may split the super-tree into a super-forest. Finally, we recurse on each component of the super-forest.

It can be seen that the total number of top tree operations needed is $\mathcal{O}(n)$. Since the top trees each contain only h vertices, a top tree operation takes $\mathcal{O}(\log h)$ time, for a total of $\mathcal{O}(n \log h)$. Detailed description and analysis of the algorithm, as well as the matching lower bound, appear in the full paper [6].

6 Conclusions

We showed that the average search time in a centroid tree is larger by at most a factor of 2 than the smallest possible average search time in an STT and that this bound is tight. We also showed that centroid trees can be computed in $\mathcal{O}(n \log h)$ time where h is the height of the centroid tree. Perhaps the most intriguing question is to determine whether the problem of computing an optimal STT is in P. A secondary goal would be to achieve an approximation ratio better than 2 in near linear time. (The running time of the STT's of Berendsohn and Kozma [7] degrade as $\mathcal{O}(n^{2k+1})$ for a $\left(1 + \frac{1}{k}\right)$-approximation.) As for centroid trees, a remaining question is whether they can be computed in $\mathcal{O}(n)$ time whenever the spread of the weight function is $\sigma \in \mathcal{O}(n)$.

A special case in which high quality approximation can be efficiently found is when an α-centroid tree exists for $\alpha < \frac{1}{2}$. This case can be recognized and handled in near linear time using our algorithm. (Observe that an α-centroid tree for $\alpha < \frac{1}{2}$ is also the unique $\frac{1}{2}$-centroid tree.) Theorem 7(ii) gives strong approximation guarantees for this case, yielding the *optimum* when $\alpha \leq \frac{1}{3}$. It is an interesting question whether the bounds can be improved for α in the range $\left(\frac{1}{3}, \frac{1}{2}\right)$, i.e., whether Theorem 7(ii) is tight.

A small gap remains in the exact approximation ratio of centroid trees when $\mathcal{T}$ has maximum degree Δ and OPT is unbounded, i.e., between the upper bound $(2 - \frac{1}{2^\Delta})$ of Theorem 3 and the lower bound $(2 - \frac{4}{2^\Delta})$ of Theorem 4(ii).

References

1 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, October 2005. doi:10.1145/1103963.1103966.

2 Bengt Aspvall and Pinar Heggernes. Finding minimum height elimination trees for interval graphs in polynomial time. *BIT*, 34:484–509, 1994.

3 Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees. *SIAM J. Comput.*, 28(6):2090–2102, 1999. doi:10.1137/S009753979731858X.

4 Michael A. Bender, Martin Farach-Colton, and Bradley C. Kuszmaul. Cache-oblivious string b-trees. In *ACM SIGMOD-SIGACT-SIGART*, pages 233–242, 2006.

5 Benjamin Aram Berendsohn. The diameter of caterpillar associahedra. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, June 27-29, 2022, Tórshavn, Faroe Islands*, volume 227 of *LIPIcs*, pages 14:1–14:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SWAT.2022.14.

6 Benjamin Aram Berendsohn, Ishay Golinsky, Haim Kaplan, and László Kozma. Fast approximation of search trees on trees with centroid trees, 2022. arXiv:2209.08024.

7 Benjamin Aram Berendsohn and László Kozma. Splay trees on trees. In *SODA*, pages 1875–1900, 2022.

8 Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. Rankings of graphs. *SIAM Journal on Discrete Mathematics*, 11(1):168–181, 1998.

9 H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

10 Prosenjit Bose, Jean Cardinal, John Iacono, Grigorios Koumoutsos, and Stefan Langerman. Competitive online search trees on trees. In *SODA*, pages 1878–1891, 2020.

11 Gerth Stølting Brodal, Rolf Fagerberg, Christian N. S. Pedersen, and Anna Östlin. The complexity of constructing evolutionary trees using experiments. In *ICALP*, pages 140–151. Springer, 2001.

12 Jean Cardinal, Stefan Langerman, and Pablo Pérez-Lantero. On the diameter of tree associahedra. *Electron. J. Comb.*, 25(4):P4.18, 2018. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v25i4p18, doi:10.37236/7762.

13 Jean Cardinal, Lionel Pournin, and Mario Valencia-Pabon. Bounds on the diameter of graph associahedra. In *Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium (LAGOS)*, volume 195 of *Procedia Computer Science*, pages 239–247. Elsevier, 2021.

14 Michael Carr and Satyan L. Devadoss. Coxeter complexes and graph-associahedra. *Topology and its Applications*, 153(12):2155–2168, 2006.

15 Cesar Ceballos, Thibault Manneville, Vincent Pilaud, and Lionel Pournin. Diameters and geodesic properties of generalizations of the associahedron. In *Proceedings of the 27th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC)*, pages 345–356, 2015.

16 Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. An almost optimal edit distance oracle. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 48:1–48:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.48.

17 Ferdinando Cicalese, Tobias Jacobs, Eduardo Laber, and Marco Molinaro. On the complexity of searching in trees and partially ordered structures. *Theor. Comput. Sci.*, 412(50):6879–6896, 2011. doi:10.1016/j.tcs.2011.08.042.

18 Ferdinando Cicalese, Tobias Jacobs, Eduardo Laber, and Marco Molinaro. Improved approximation algorithms for the average-case tree searching problem. *Algorithmica*, 68(4):1045–1074, 2014. doi:10.1007/s00453-012-9715-6.

19 Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality - almost. *SIAM J. Comput.*, 37(1):240–251, 2007. doi:10.1137/S0097539705447347.

20 Jitender S Deogun, Ton Kloks, Dieter Kratsch, and Haiko Müller. On vertex ranking for permutation and other graphs. In *STACS 1994*, pages 747–758. Springer, 1994.

21 Satyan L. Devadoss. A realization of graph associahedra. *Discrete Mathematics*, 309(1):271–276, 2009.

22 Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices.* Oxford University Press, 2017.

23 Guy Even and Shakhar Smorodinsky. Hitting sets online and unique-max coloring. *Discret. Appl. Math.*, 178:71–82, 2014. doi:10.1016/j.dam.2014.06.019.

24 Paolo Ferragina. On the weak prefix-search problem. *Theor. Comput. Sci.*, 483:75–84, 2013. doi:10.1016/j.tcs.2012.06.011.

25 Paolo Ferragina and Rossano Venturini. Compressed cache-oblivious string b-tree. *ACM Trans. Algorithms*, 12(4):52:1–52:17, 2016. doi:10.1145/2903141.

26 Greg N. Frederickson and Donald B Johnson. Finding kth paths and p-centers by generating and searching good data structures. *Journal of Algorithms*, 4(1):61–80, 1983.

27 Travis Gagie, Danny Hermelin, Gad M Landau, and Oren Weimann. Binary jumbled pattern matching on trees and tree-like structures. *Algorithmica*, 73(3):571–588, 2015.

28 Davide Della Giustina, Nicola Prezza, and Rossano Venturini. A new linear-time algorithm for centroid decomposition. In *Proceedings of the 26th International Symposium on String Processing and Information Retrieval (SPIRE)*, volume 11811 of *Lecture Notes in Computer Science*, pages 274–282. Springer, 2019.

29 Michael T. Goodrich and Roberto Tamassia. Dynamic trees and dynamic point location. *SIAM J. Comput.*, 28(2):612–636, 1998. `doi:10.1137/S0097539793254376`.

30 Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987. `doi:10.1007/BF01840360`.

31 Brent Heeringa, Marius Catalin Iordan, and Louis Theran. Searching in dynamic tree-like partial orders. In *WADS 2011*, volume 6844 of *Lecture Notes in Computer Science*, pages 512–523. Springer, 2011. `doi:10.1007/978-3-642-22300-6_43`.

32 D.S. Hirschberg, L.L. Larmore, and M. Molodowitch. Subtree weight ratios for optimal binary search trees. Technical Report TR 86-02, ICS Department, University of California, Irvine, 1986.

33 Ananth V. Iyer, H. Donald Ratliff, and Gopalakrishnan Vijayan. Optimal node ranking of trees. *Inf. Process. Lett.*, 28(5):225–229, 1988. `doi:10.1016/0020-0190(88)90194-9`.

34 Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

35 Meir Katchalski, William McCuaig, and Suzanne Seager. Ordered colourings. *Discrete Mathematics*, 142(1-3):141–154, 1995.

36 Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, 1971. `doi:10.1007/BF00264289`.

37 Tomasz Kociumaka, Jakub Pachocki, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Efficient counting of square substrings in a tree. *Theoretical Computer Science*, 544:60–73, 2014.

38 Eduardo Laber and Marco Molinaro. An approximation algorithm for binary searching in trees. *Algorithmica*, 59(4):601–620, 2011. `doi:10.1007/s00453-009-9325-0`.

39 Eduardo Laber and Loana Nogueira. Fast searching in trees. *Electronic Notes in Discrete Mathematics*, 7:90–93, 2001. `doi:10.1016/S1571-0653(04)00232-X`.

40 Lawrence L. Larmore. A subquadratic algorithm for constructing approximately optimal binary search trees. *J. Algorithms*, 8(4):579–591, 1987. `doi:10.1016/0196-6774(87)90052-6`.

41 Charles E. Leiserson. Area-efficient graph layouts (for VLSI). In *STOC 1980*, pages 270–281. IEEE Computer Society, 1980. `doi:10.1109/SFCS.1980.13`.

42 Nathan Linial and Michael E. Saks. Every poset has a central element. *J. Comb. Theory, Ser. A*, 40(2):195–210, 1985. `doi:10.1016/0097-3165(85)90087-1`.

43 Joseph W.H. Liu. The role of elimination trees in sparse factorization. *SIAM journal on matrix analysis and applications*, 11(1):134–172, 1990.

44 Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, 1975.

45 Kurt Mehlhorn. A best possible bound for the weighted path length of binary search trees. *SIAM Journal on Computing*, pages 235–239, 1977.

46 Shay Mozes, Krzysztof Onak, and Oren Weimann. Finding an optimal tree searching strategy in linear time. In *SODA 2008*, pages 1096–1105. SIAM, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347202`.

47 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. `doi:10.1007/978-3-642-27875-4`.

48 Krzysztof Onak and Pawel Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *FOCS 2006*, pages 379–388, 2006. `doi:10.1109/FOCS.2006.32`.

49 Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM journal on matrix analysis and applications*, 11(3):430–452, 1990.

50 Alejandro A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2):91–96, 1989.

51 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985. doi:10.1145/3828.3835.