Better Approximations for Tree Sparsity in Nearly-Linear Time

Arturs Backurs
MIT

Piotr Indyk
MIT

Ludwig Schmidt
MIT

Abstract

The Tree Sparsity problem is defined as follows: given a node-weighted tree of size n and an integer k, output a rooted subtree of size k with maximum weight. The best known algorithm solves this problem in time $O(kn)$, i.e., quadratic in the size of the input tree for $k = \Theta(n)$.

In this work, we design $(1+\varepsilon)$-approximation algorithms for the Tree Sparsity problem that run in nearly-linear time. Unlike prior algorithms for this problem, our results offer single criterion approximations, i.e., they do not increase the sparsity of the output solution, and work for arbitrary trees (not only balanced trees). We also provide further algorithms for this problem with different runtime vs approximation trade-offs.

Finally, we show that if the exact version of the Tree Sparsity problem can be solved in strongly subquadratic time, then the $(\min, +)$ convolution problem can be solved in strongly subquadratic time as well. The latter is a well-studied problem for which no strongly subquadratic time algorithm is known.

1 Introduction

The *tree sparsity* problem is defined as follows: given a node-weighted tree of size n and an integer k, output a rooted subtree of size k with the maximum weight. Over the last two decades, this problem and its variants have been the subject of extensive studies in several areas. In combinatorial optimization, this problem constitutes an important special case of the well-studied rooted k-MST problem[1] which is NP-hard in general, but solvable on trees in polynomial time [RSM$^+$96, LZ93]. In machine learning, the problem was formulated in the context of optimal decision tree pruning [BB94]. Perhaps most prominently, the problem and its variants have been studied in the sparse approximation and compressive sensing literature, where it is used to identify tree sparse approximations of signals represented in a wavelet basis (see e.g., [BJ93, Don97, BCDH10, CT13, HIS14b] as well as the recent survey [HIS15b]). In the latter two applications (machine learning and sparse recovery), the trees are often assumed to be binary. This is the case we consider in the rest of this paper.

The fastest exact algorithm for this problem runs in time $O(kn)$ [CT13].[2] Since this running time can be quite high[3] for large values of k and n, multiple faster but approximate or heuristic algorithms

[1]In the rooted k-MST problem we are given an edge-weighted graph G, the size parameter k, and a node v. The goal is to find a subtree of G with k edges that contains v and has the minimum weight. Note that for the case when G is a tree, rooted k-MST is computationally equivalent to tree sparsity: the weight of each edge can be assigned to one of its endpoints that points "away" from the root v, and minimization can be replaced by maximization by negating each weight and adding to it the sum of all weights W.

[2]The algorithm in [CT13] was designed for *balanced* binary trees. However, in the appendix we show the algorithm can extended to arbitrary binary trees.

[3]E.g., for mega-pixel images we could have $n \approx 10^6$ and $k \approx 0.1n \approx 10^5$.

were designed [Don97, BCDH10, CT13, HIS14b]. The approximate variants of the problem can be formulated in two *different* ways.[4] The first one relaxes the *maximization* problem, where the goal is to find a sub-tree whose weight is as large as possible. The second one relaxes the *minimization* problem, where the goal is to find a subtree such that the sum of node weights *not* in the subtree is as small as possible. In both cases, the approximation algorithms find solutions that are within a constant factor away from the optimum. Furthermore, the algorithms output sub-trees of size that is larger than k by a constant factor. Table 1 summarizes the details of these algorithms (only the best known bounds are shown). Note that the prior algorithms listed in the table work only for balanced trees, i.e., of depth $O(\log n)$.

Reference	Max/Min	Running time	Solution sparsity	Approximation ratio	Tree depth
[CT13]	Either	$O(kn)$	k	1	$O(\log n)$
[HIS14b]	Min	$O(n \log n)$	$2k$	≤ 2	$O(\log n)$
[HIS14b]	Max	$O(n \log n + k \log^2 n)$	$2k + O(\log n)$	$\geq 1/4$	$O(\log n)$
This paper	Min	$n(\log n + 1/\epsilon)^{O(1)}$	k	$\leq (1+\epsilon)$	any
This paper	Max	$n(\log n + 1/\epsilon)^{O(1)}$	k	$\geq (1-\epsilon)$	any
This paper	Max	$O(n \log n)$	k	$\geq (k-d)/16k$	d

Table 1: A summary of the algorithms for the tree sparsity problem. Only the best known bounds are included. The algorithms in [HIS14b] offer tradeoffs between sparsity and approximation factors that we do not show for simplicity. We distinguish between the minimization and the maximization versions of the problem. The approximation ratio is relative to the value of the optimal solution.

Overall, the existing algorithms are either (i) exact but slow, or (ii) fast but approximate, yielding solutions whose value and sparsity bounds are suboptimal by constant factors. In particular, no constant factor algorithm with near-linear running time was known for the minimization variant of the problem if the sparsity bound k cannot be exceeded. The existence of such an algorithm was posed by the last two authors of this paper as an open problem ([HIS15b], Problem 2). In the same survey, they asked whether the running time of the exact algorithm can be improved.

Our results In this paper, we resolve the first question and make progress on the latter. Our main results are as follows:

- We present nearly linear-time *single-criterion* approximation algorithms for both the maximization and the minimization variants. Furthermore, the approximation factors of our algorithms are arbitrarily close to 1.
- We show that, for $k \approx n$, a strongly sub-quadratic time algorithm for tree sparsity would imply a strongly sub-quadratic time algorithm for $(\min, +)$-*convolution*. The latter is a well-studied problem (see e.g., [BCD$^+$06]) for which no strongly sub-quadratic time algorithm is known. The problem bears strong similarity to the $(\min, +)$-matrix product, which has been used as the basis for several conditional hardness results [RZ04, WW10, AW14, AGW15, AVWY15, BDT16]. In particular, our reduction uses the techniques of [WW10].

[4]The choice of the formulation depends on the application. In particular, the application to approximation-tolerant model-based compressive sensing described in [HIS15a] requires *both* variants. See [HIS15b] for further discussion.

We also give a simple algorithm for the maximization variant whose running time $O(n \log n)$ has only a single logarithmic factor. The approximation factor of the algorithm is constant for trees whose depth is a constant factor smaller than the sparsity k. We also show that a "boosted" variant of the algorithm increases the sparsity bound by a logarithmic factor while achieving an approximation ratio of 1.

Our techniques Our algorithms and lower bounds exploit the close connection between tree sparsity and convolutions. This can be illustrated by recalling the $O(kn)$-time exact algorithm for tree sparsity. The algorithm is based on dynamic programming. Specifically, for each node i and sparsity t, we compute $W[i,t]$, defined as the value of a sub-tree of size t, rooted at node i, and maximizing the total weight of the selected nodes. We compute this quantity recursively by selecting i and then, for an appropriate parameter r, selecting the maximum weight subtree of size r rooted in the left child of i, as well as the maximum weight subtree of size $t-r-1$ rooted in the right child of i. Since we do not know the optimal value of r in advance, the algorithm enumerates all possible values. This leads to the following recursive formula:

$$W[i,t] = x_i + \max_r W[\mathit{left}(i), r] + W[\mathit{right}(i), t-r-1] \tag{1}$$

where x_i denotes the weight of the node i.

Equation 1, when executed for all $t = 0 \ldots k$, can be seen as corresponding to the $(\max,+)$ convolution of a sequence

$$W[\mathit{left}(i), 0] \ldots W[\mathit{left}(i), t-1]$$

and

$$W[\mathit{right}(i), 0] \ldots W[\mathit{right}(i), t-1]$$

Our lower bound proceeds by showing that this correspondence is in some sense inherent, as we reduce $(\max,+)$ convolution to tree sparsity over a binary tree of depth $\Theta(k)$ with three long paths from the root to the leaves. In turn, our approximation algorithms are obtained by solving *approximate* $(\max,+)$ convolutions or $(\min,+)$-convolutions. Such approximations are known to be computable in nearly-linear time [Zwi98, PS16]. For completeness, we include simpler (albeit slightly slower) algorithms performing those tasks in the appendix.

The first step in our approximation algorithms is to replace the exact convolution in Equation 1 by its approximate counterparts. However, this works only if the underlying tree has bounded (say, polylogarithmic) depth. This is because the approximation errors incurred at each level are cumulative, i.e., the final error bound is exponential in the tree depth. In order to make the algorithms applicable for trees of arbitrary depth, we must ensure that the error accumulates only a limited number of times. To this end, we utilize the *heavy-light decomposition* [ST83] of the underlying tree. On a high level, this approach decomposes the tree into a set of (possibly long) paths called *spines*. The decomposition ensures that there are at most $O(\log n)$ such spines on a path from the root to any leaf. We then design an algorithm that processes each spine in "one shot", as opposed to node by node. In this way, we ensure that the error is accumulated only a poly-logarithmic number of times.

2 Preliminaries

We start by setting up basic notation. We generally identify the node weights of a given binary tree with an n-dimensional vector $x \in \mathbb{R}^n$: each of the n coefficients in x corresponds to a node in the tree. Similarly, we identify a subtree with the corresponding *support* $\Omega \subseteq [n]$ in the vector x, i.e., the set of indices belonging to the subtree. We denote the set of supports forming valid rooted subtrees of size k with $\mathbb{T}_k$. For an arbitrary vector / tree x, we write x_Ω for the restriction of x to the support / subtree Ω: we have $(x_\Omega)_i = x_i$ if $i \in \Omega$ and $(x_\Omega) = 0$ otherwise. For a given support Ω, we denote its complement with $\overline{\Omega} = [n] \setminus \Omega$. Finally, the ℓ_1-norm of a vector x is $\|x\|_1 = \sum_{i=1}^n \|x_i\|$.

2.1 Head and tail approximation

With the new notation in place, we now formally state our main algorithmic problems. The "head" and "tail" terminology comes from the application of tree sparsity in sparse recovery [HIS15a]. In the following problems, the input is a tree with node weights $x \in \mathbb{R}^n$ and a sparsity parameter k.

Exact tree sparsity Find a subtree $\Omega \in \mathbb{T}_k$ such that $\|x - x_\Omega\|_1$ is minimized. Note that this is equivalent to maximizing $\|x_\Omega\|_1$.

Tail approximation Find a subtree $\Omega \in \mathbb{T}_k$ such that $\|x - x_\Omega\|_1 \leq c_T \cdot \min_{\Omega' \in \mathbb{T}_k} \|x - x_{\Omega'}\|_1$, where c_T is a constant (ideally close to 1).

Head approximation Find a subtree $\Omega \in \mathbb{T}_k$ such that $\|x_\Omega\|_1 \geq c_H \cdot \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1$, where c_H is a constant (ideally close to 1).

The tail approximation problem can be seen as an approximate minimization problem, while the head approximation problem is an approximate maximization problem. Note that a constant-factor tail approximation does not imply a constant-factor head approximation, and vice versa.

2.2 Generalized convolutions

Our algorithms build on sub-routines for various convolution problems, which we now review.

Definition 1 (($\otimes, \oplus$)-Convolution problem)**.** *Given two vectors* $A = (A_0, \ldots, A_{n-1})^T \in \mathbb{Z}^n$ *and* $B = (B_0, \ldots, B_{n-1})^T \in \mathbb{Z}^n$*, output the vector* $C = (C_0, \ldots, C_{n-1})^T \in \mathbb{Z}^n$ *defined as:*

$$C_k = \otimes_{i=0}^{k} (A_i \oplus B_{k-i}) \qquad \text{for all } k = 0, \ldots, n-1 \; .$$

We consider three instances of the general convolution problem: $(\min, +)$, $(\max, +)$, and (OR, AND)-convolution. The latter is also called *Boolean Convolution.* Note that one can solve the Boolean Convolution problem in time $O(n \log n)$ using the Fast Fourier Transform on indicator vectors.

We will repeatedly use the following two theorems about $(\min, +)$ and $(\max, +)$-convolutions that we prove in Appendix C.

Let $A, B \in \mathbb{N}^n$ be two integer vectors with positive entries. Let W be the largest value of an entry in A or B. Let $C \in \mathbb{N}^n$ be the $(\max, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1 + \varepsilon) C'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W\right) .$$

An analogous statement holds if C is the $(\min, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1 + \varepsilon) C'_k$ for all $k = 0, \ldots, n-1$ in the same runtime.

An slightly faster algorithm for $(\min, +)$ matrix products was shown by Uri Zwick [Zwi98]. We also note than a different algorithm for approximating $(\max, +)$-convolution was given in [Ser15], but no theoretical guarantees on the approximation factor were provided.

Let $A^1 \in \mathbb{N}^{n_1}, \ldots, A^l \in \mathbb{N}^{n_l}$ be $l \geq 2$ vectors with positive entries. We assume that the entries of the vectors are indexed starting with 0, i.e., $A^i = \left(A^i_0, \ldots, A^i_{n_i - 1}\right)^T$ for every $i = 1, \ldots, l$. Let $B \in \mathbb{N}^n$, $n = n_1 + \ldots + n_l$ be the $(\min, +)$-convolution between the l vectors $A^1, \ldots, A^{n_l}$ defined as follows:

$$B_j = \min_{\substack{m_1 + \ldots + m_l = j \\ \text{s.t. } 0 \leq m_t \leq n_t - 1 \ \forall t = 1, \ldots, l}} \left(A^1_{m_1} + \ldots + A^l_{m_l}\right)$$

for every $j = 0, \ldots, n-1$. We can output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W \cdot \log^3 l\right),$$

where W denotes the largest entry in the vectors $A^1, \ldots, A^l$.

An analogous statement holds for $(\max, +)$-convolution (we replace min in the equation for B_j with max). The runtime stays the same and we output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$.

3 $1 + \varepsilon$ tail approximation in nearly linear time

Our algorithm utilizes a heavy-light decomposition [ST83], which we define first.

Definition 2 (Heavy-light decomposition [ST83])**.** *Let x be a binary tree, and let node v be a child of node u (the parent). We call an edge $e = (u, v)$* heavy *if one of the following holds:*

- $\mathit{size}(v) > \frac{\mathit{size}(u) - 1}{2}$;
- $\mathit{size}(v) = \frac{\mathit{size}(u) - 1}{2}$ *and v is the right child of the parent u.*

We call all other edges light.

We extend the above definition of light and heavy edges to nodes as follows: If a parent u is connected to a child v by a heavy edge, we call v heavy. We can easily check that every node has at most one heavy child and that only leaves can have no heavy child. We call a node *special* if it is not a heavy child of its parent. Special nodes are relevant for our definition of a *spine*:

Definition 3 (Spine)**.** *Consider the following process. Pick any special node u. If u is not a leaf, it has a unique heavy child. Choose this child and repeat. We will end up in a leaf. This defines a path in the tree corresponding to the node u. We call this path the spine corresponding to the node u. The spine includes the node u and the final leaf.*

By iterating over all special nodes, we can decompose the tree into a set of spines. This decomposition has desirable properties because we can only have a small number of spines on any path from a leaf to the root. More formally, consider any node u in the tree. The number of light edges on the path from the node u to the root of the tree is upper bounded by $O(\log n)$. The reason is that every light edge at least halves the size of the subtree and the number of nodes in the tree is upper bounded by n. Since every two light edges on the path are separated by one spine, the number of different spines on the path from the node u to the root is also upper bounded by $O(\log n)$.

With our definition of a spine set up, we now proceed to our main result of this section.

Theorem 4. *Given a vector x and an integer $k \geq 1$, we can find a subtree $\Omega \in \mathbb{T}_k$ such that*

$$\|x_{\overline{\Omega}}\|_1 \; \leq \; (1+\varepsilon) \cdot \|x_{\overline{\Omega_k^*}}\|_1$$

where

$$\Omega_k^* \; = \; \operatorname*{arg\,min}_{\Omega' \in \mathbb{T}_k} \|x_{\overline{\Omega'}}\|_1 \; .$$

Moreover, the algorithm runs in time $O(\frac{n}{\varepsilon^3} \cdot \log^9 n \cdot \log^3 x_{\max})$.

Proof. For a tree x of size $n \geq 0$, we define the *tail tree sparsity vector* to be the vector

$$\left(\|x_{\overline{\Omega_0^*}}\|_1, \|x_{\overline{\Omega_1^*}}\|_1, \ldots, \|x_{\overline{\Omega_n^*}}\|_1\right)^T \in \mathbb{N}^{n+1}.$$

To simplify the exposition, in this section we will skip the word "tail".

We will recursively compute approximate tree sparsity vectors for all subtrees rooted at the special nodes of the tree x. As we compute these approximations, the sizes of the subtrees become smaller and the lengths of the corresponding approximate sparsity vectors (the number of entries in the vectors) also become smaller. To compute the approximate sparsity vector for a special note u, we need to know the approximate sparsity vectors for all special nodes in the subtree rooted at u. Without loss of generality, we assume that for every leaf l, we have $x_l > 0$.

Our recursive algorithm works as follows.

- For an integer $l \geq 1$, let $p_1, p_2, \ldots, p_l$ be the spine that starts at the root p_1 of the tree x and ends in the leaf p_l of the tree. For simplicity, we renumber vertices of the tree x so that the spine consists of vertices $1, \ldots, l$ in this order (1 is the root of the tree and l is the leaf).

- For $i = 1, \ldots, l-1$, the node i has $i+1$ as a child. Let x^i denote the subtree rooted at the other child node (recall that x is a binary tree). Let $n_i \geq 0$ denote the size of the subtree x^i. Note that n_i can be zero if node i has only one child, i.e., the node $i+1$. For simplicity of exposition, let x^l be an empty tree of size $n_l := 0$. It corresponds to a child of leaf l. Since leaf l has no children, x^l is of size 0. Let

$$s^i := \left(s_0^i, s_1^i, s_2^i, \ldots, s_{n_i}^i\right)^T \in \mathbb{N}^{n_i+1}$$

  be the approximate sparsity vector for the subtree x^i computed recursively. We assume that s^i is an approximate sparsity vector in the following sense. Let $\hat{s}^i$ be the (exact) sparsity vector for the subtree x^i. Then $\hat{s}_j^i \leq s_j^i \leq (1+\delta)\hat{s}_j^i$ for some $\delta > 0$ and for all $j = 0, 1, \ldots, n_i$.

- For a node $i = 1, \ldots, l$, let w_i denote the total weight of the subtree rooted at i. We define a set L as follows. Initially $L = \{l\}$. We set $l_{\min} = l$. While $l_{\min} > 1$, choose the smallest $l' \geq 1$ such that $w_{l'+1} \leq (1+\delta')w_{l_{\min}}$ for some $\delta' > 0$ that we will choose later. Then we add l' to L, set $l_{\min} = l'$, and repeat. We note that the size of the set L is upper bounded by $|L| \leq \log_{1+\delta'}(nx_{\max}) \leq O(\log(nx_{\max})/\delta')$.

  We need the set L for the following reason: We want to approximate the sparsity vector for the tree x. Fix an arbitrary sparsity $t \geq 1$ that we want to approximate. The optimal tree will pick a number $l'' \geq 1$ of nodes from the spine and rooted subtrees from the trees $x^1, \ldots, x^{l''}$. The main idea is that l'' is as good as l''' for some $l''' \in L$ up to a factor $(1+\delta')$. That is, we can assume that $l'' \in L$ and we loose at most a factor of $(1+\delta')$ in the approximation ratio (we make this precise in the analysis below). This implies that it is sufficient to compute the sparsity vectors for all different $l'' \in L$, i.e., assuming that we have to pick the first l'' nodes from the spine. In particular, we do *not* require to compute sparsity vectors for all values in $[l]$, which leads to an important speed-up and enables our algorithm to run in nearly-linear time. To get an approximation for sparsity t, we take the minimum tail (for sparsity t) that we achieve over all the computed sparsity vectors for different $l'' \in L$.

- For every $l'' \in L$, we compute a sparsity vector $r^{l''} \in \mathbb{N}^{n+1}$. Let t be the sparsity that we want to approximate. We want to pick up the first l'' nodes from the spine. We also have to pick up $t - l''$ nodes from the trees $x^1, \ldots, x^{l''}$. Therefore, we set

$$r_t^{l''} := x_1 + \ldots + x_{l''} + \min_{\substack{m_1,\ldots,m_{l''} \geq 0 \\ \text{s.t. } m_1+\ldots+m_{l''}=t-l''}} \left(s^1_{m_1} + \ldots + s^{l''}_{m_{l''}} \right).$$

  It remains to approximately compute $\min\left(s^1_{m_1} + \ldots + s^{l''}_{m_{l''}}\right)$. This is exactly the problem stated in Theorem 2.2. We run the corresponding algorithm with approximation factor $(1+\delta')$. The approximate sparsity vector $r \in \mathbb{N}^{n+1}$ for the tree x is then computed as $r_t := \min_{l'' \in L} r_t^{l''}$ for all $t = 0, 1, \ldots, n$.

Correctness of the algorithm There are three sources of error in one recursive call of the algorithm. First, we do not have the exact sparsity vectors for the subtrees x^i. Instead, we have $(1+\delta)$-approximate sparsity vectors. This introduces $1+\delta$ multiplicative error in our approximation of the sparsity vector for the tree x. Second, we will show below that working with the prefixes of the spine of length $l'' \in L$ introduces a multiplicative error of $(1+\delta')$. Finally, since we perform an *approximate* $(\min,+)$-convolution in the final step of the recursive call, we get additional multiplicative error of $(1+\delta')$. Therefore, the multiplicative step gives an approximate sparsity vector for the tree x with error $(1+\delta)(1+\delta')^2$. From Definition 2, we know that the number of different spines on any path to the root is upper bounded by $O(\log n)$. This implies that the recursive depth of the algorithm is $O(\log n)$, which leads to a final approximation error of $(1+\delta')^{O(\log n)}$. Choosing $\delta' = \Theta\left(\frac{\varepsilon}{\log n}\right)$ gives the promised $(1+\varepsilon)$ approximation factor.

It remains to show that working with the prefixes of the spine of length $l'' \in L$ introduces a multiplicative error of at most $(1+\delta')$. Fix an arbitrary sparsity $t = 0, \ldots, n$. Let l' be the number of nodes from the spine that an optimal subtree (with the smallest possible tail error) of size t picks up. We assume that $l' \notin L$ since otherwise we compute an exact (sub-)solution. We have that $l''_1 < l' < l''_2$ for some $l''_1, l''_2 \in L$ with $w_{l''_1+1} \leq (1+\delta') w_{l''_2}$ by the construction of the set L. Let Ω^* be the support of the optimal subtree (Ω^* picks up l' nodes from the spine). Since $l' < l''_2$, we have

$$w_{l''_2} \leq \|x_{\overline{\Omega^*}}\|_1. \tag{2}$$

Let Ω_t be Ω^* but with all the nodes from the subtree rooted at $l''_1 + 1$ removed from it. Then Ω_t picks up l''_1 nodes from the spine as required. We have to show that we did not increase the tail too much by removing vertices from the support Ω^* to get the support Ω_t. To this end, we observe that we increased the tail by at most $w_{l_1''+1} - w_{l_2''}$, and therefore:

$$\begin{aligned}\|x_{\overline{\Omega_t}}\|_1 &\leq \|x_{\overline{\Omega^*}}\|_1 + w_{l_1''+1} - w_{l_2''} \\ &\leq \|x_{\overline{\Omega^*}}\|_1 + \delta' w_{l_2''} \\ &\leq (1+\delta')\|x_{\overline{\Omega^*}}\|_1,\end{aligned}$$

where we use (2) in the last inequality.

The runtime of the algorithm There are $S_1 := O(\log n)$ levels of recursion in the algorithm. In each level, we perform $|L| \leq O(\log(nx_{\max})/\delta') =: S_2$ approximate $(\min, +)$-convolutions between multiple sequences with approximation factor $(1+\delta')$. From Theorem 2.2, we know that one $(\min, +)$-convolution takes $S_3 := O\left(\frac{n}{\delta'^2} \cdot \log^4 n \cdot \log^2 x_{\max}\right)$ time. Since we chose $\delta' = \left(\frac{\varepsilon}{\log n}\right)$, we get the final runtime stated in the theorem:

$$S_1 \cdot S_2 \cdot S_3 \leq O\left(\frac{n}{\varepsilon^3} \cdot \log^9 n \cdot \log^3 x_{\max}\right)$$

□

4 $1+\varepsilon$ head approximation in nearly linear time

In this section, we prove the following theorem.

Theorem 5. *Given a vector x and an integer $k \geq 1$, we can find a subtree $\Omega \in \mathbb{T}_k$ such that*

$$\|x_\Omega\|_1 \ \geq \ \|x_{\Omega_k^*}\|_1/(1+\varepsilon)$$

where

$$\Omega_k^* \ = \ \arg\max_{\Omega_k \in \mathbb{T}_k} \|x_{\Omega_k}\|_1 \ .$$

Moreover, the algorithm runs in time $O(\frac{n}{\varepsilon^2} \cdot \log^{12} n \cdot \log^2 x_{\max})$.

We solve the problem in two attempts. In the first attempt, we construct an algorithm that runs in the required nearly-linear time but the approximation error is too large. This will allow us to introduce ideas used in the second (and final) algorithm that achieves both the desired running time and approximation factor.

4.1 First attempt to solve the problem

For a tree x of size $n \geq 0$, we define the *head tree sparsity vector* to be the vector

$$\left(\|x_{\Omega_1^*}\|_1, \ldots, \|x_{\Omega_n^*}\|_1\right)^T \in \mathbb{N}^n.$$

Notice that this definition is different from the *tail* tree sparsity vector in Section 3. To simplify the exposition, we omit the word "head".

Our plan is to compute an approximation to the tree sparsity vector for every tree rooted at a special node, i.e., for every tree rooted at the first node of every spine. Since the root of the tree is a special node, this will also solve our original problem.

Similarly to the previous section, we perform this computation recursively.

- For an integer $l \geq 1$, let $p_1, p_2, \ldots, p_l$ be the spine that starts at the root p_1 of the tree x and ends in a leaf (p_l) of the tree. To simplify notation, we renumber the vertices of the tree x so that the spine consists of vertices $1, \ldots, l$ in this order.

- For $i = 1, \ldots, l-1$, node i has child node $i+1$. Let x^i denote the subtree rooted at the other child node. Let $n_i \geq 0$ denote the size of the subtree x^i. To simplify notation, let x^l be an empty tree of size $n_l := 0$. It corresponds to a child of leaf l. Let

$$s^i := \left(s^i_1, s^i_2, \ldots, s^i_{n_i}\right)^T \in \mathbb{N}^{n_i}$$

  be the approximate sparsity vector for the subtree x^i computed recursively. We assume that s^i is a $(1+\delta)$-approximate sparsity vector in the following sense. Let $\hat{s}^i$ be the (exact) sparsity vector for the subtree x^i. Then $\hat{s}^i_j/(1+\delta) \leq s^i_j \leq \hat{s}^i_j$ for all $j = 1, \ldots, n_i$. We will determine $\delta > 0$ later.

- Let $t = (t_1, \ldots, t_m)^T \in \mathbb{N}^m$ be the approximate sparsity vector corresponding to a tree of size $m \geq 0$. We would like to represent t with much fewer than m entries so that we can work with the sparsity vector faster. For this, we define the *compressed sparsity vector* $C(t) \in \mathbb{N}^{\log_{1+\delta'}(mx_{\max})}$ as follows.[5] For $j = 0, \ldots, \log_{1+\delta'}(mx_{\max}) - 1$, we set $C(t)_j$ to the minimum $j' \geq 1$ such that $t_{j'} \geq (1+\delta')^j$. Notice that the sparsity vector is non-decreasing (that is, $t_k \leq t_{k+1}$ for every $k = 1, \ldots, m-1$). Intuitively, $C(t)$ stores the indices of the sparsity vector t where the value changes by a lot, i.e., by at least a factor of $1+\delta'$. We decreased the number of entries from m to $\log_{1+\delta'}(mx_{\max}) \leq O(\log(mx_{\max})/\delta')$ by using $C(t)$ instead of t. Note that, given $C(t)$, we can recover vector $r = (r_1, \ldots, r_m)^T \in \mathbb{N}^m$ such that $t_j/(1+\delta') \leq r_j \leq t_j$ for every $j \in [m]$. We do this as follows: we set $r_j := \max_{j' : C(t)_{j'} \leq j} (1+\delta')^{j'}$ for every $j \in [m]$.

- For every $i \in 1 \ldots l$, we compute the compressed sparsity vector $C(s^i)$ corresponding to the subtree x^i. Then, for every $i = l, l-1, \ldots, 1$ (in this order), we compute the compressed sparsity vector c^i for the tree rooted at node i. To compute c^i we need $C(s^i)$ and c^{i-1}. Given $C(s^i)$ and c^{i-1}, we can first compute approximate sparsity vectors corresponding to the tree x^i and to the tree rooted at node $i-1$ as described in the definition of compressed sparsity vectors above. Then we could use $(\max, +)$-convolution to compute the approximate sparsity pattern for the tree rooted at node i from which we can obtain c^i. However, the time complexity of this approach is too large. Instead, given c^{i-1} and $C(s^i)$, we can compute c^i directly in time $O(\log^2(nx_{\max})/\delta'^2)$. We give more details below.

  We output c^1, which is the compressed sparsity vector corresponding to the input tree x.

Why this approach does not quite work Consider the stage of the algorithm when we are given compressed sparsity vectors c^{i-1} and $C(s^i)$ and we compute c^i for some fixed $i = 1, \ldots, l$. To compute c^i, we can (implicitly) compute an approximate sparsity vector a corresponding to c^{i-1} and an approximate sparsity vector b corresponding to $C(s^i)$ as described in the definition of the compressed sparsity vectors. The vectors a and b consists of at most $O(\log(nx_{\max})/\delta')$ different entries. It is not hard to see that the approximate sparsity vector for the tree rooted at node i consists of at most $O(\log^2(nx_{\max})/\delta'^2)$ different values (it is essentially a $(\max, +)$-product of a and b). We obtain c^i from the resulting approximate sparsity vector as in the definition of the

[5] We will choose $\delta' > 0$ later. $x_{\max}$ upper bounds the maximum value in the vector t.

compressed sparsity vectors. Again, it is easy to check that this can be done implicitly in the stated runtime. As observed in the definition, the step of obtaining the compressed sparsity vector c^i introduces a $1+\delta'$ multiplicative error because we round the values of the vector to an integer power of $1+\delta'$. Since we compute l compressed sparsity vectors $c^l, c^{l-1}, \ldots, c^1$, the total error that we collect is $(1+\delta')^l$. Since we want the final error to be small (at most $1+\varepsilon$), and l can be large (as large as $\Omega(n)$), we have to choose $\delta' = O(1/n)$. This is prohibitively small because even the size of the compressed sparsity vectors becomes $\Omega(n)$.

4.2 Second attempt to solve the problem

In our first attempt, we constructed an algorithm in which we had to choose δ' to be too very small. In this section, we change the algorithm in a way that will allow us to pick δ' to be much larger. The resulting algorithm then runs in the promised time complexity and achieves a $1+\varepsilon$ approximation. The algorithm stays the same as in attempt one, except for the last (fourth) step. We now describe how to modify this last step.

W.l.o.g. we assume that l is an integer power of 2. We can do that since otherwise we can add $l - 2^j\lfloor l/2^j \rfloor$ nodes to the spine with the corresponding values equal to 0. For every integer $j \geq 0$ such that $2^j \leq l$, we split the nodes $1, 2, \ldots, l$ on the spine into $l/2^j$ groups, each containing 2^j nodes. For $y = 1, \ldots, l/2^j$, the y-th group consists of 2^j nodes $(y-1)2^j+1, (y-1)2^j+2, \ldots, y2^j$. For a fixed $j \geq 0$ and for a fixed group y, we want to compute an approximate sparsity vector $r^{j,y}$ corresponding to subtrees in which we choose only nodes $(y-1)2^j+1, (y-1)2^j+2, \ldots, y2^j$ from the spine. Namely, we want to compute

$$\begin{aligned} r_t^{j,y} :=& x_{(y-1)2^j+1} + \ldots + x_{y2^j} \\ &+ \max_{\substack{m_1,\ldots,m_{2^j} \geq 0 \\ \text{s.t. } m_1+\ldots+m_{2^j} = t-2^j}} \left(s_{m_1}^{(y-1)2^j+1} + \ldots + s_{m_{2^j}}^{y2^j} \right). \end{aligned}$$

for all $t = 0, 1, \ldots, 2^j + n_{(y-1)2^j+1} + n_{(y-1)2^j+2} + \ldots + n_{y2^j}$. To approximately compute the quantity $\max\left(s_{m_1}^{(y-1)2^j+1} + \ldots + s_{m_{2^j}}^{y2^j}\right)$, we run the algorithm from Theorem 2.2 with approximation factor $1+\delta'$. Let $n^{j,y} := 2^j + n_{(y-1)2^j+1} + \ldots + n_{y2^j}$. The runtime for $(1+\delta')$-approximately computing $r^{j,y}$ is

$$O\left(\frac{n^{j,y}}{\delta'^2} \cdot \log^6 n \cdot \log^2 x_{\max}\right)$$

by Theorem 2.2. When using Theorem 2.2 we note that the largest value in vectors s^i can be $\Omega(nW)$. Since $n^{j,1} + \ldots + n^{j,l/2^j} \leq n$, we get that the total runtime for computing the approximations $\hat{r}^{j,y}$ for all vectors $r^{j,y}$ is upper bounded by

$$O\left(\frac{n}{\delta'^2} \cdot \log^7 n \cdot \log^2 x_{\max}\right). \tag{3}$$

Now we will describe how to use the vectors $\hat{r}^{j,y}$ to get an approximation to the sparsity vector of the tree x. We start with computing the compressed sparsity vectors $C(\hat{r}^{j,y})$ for all j and y.

For every $i = 1, 2, \ldots, l$, we consider trees in which we choose nodes $1, 2, \ldots, i$ from the spine and we do not choose the node $i+1$. For such trees, we compute the compressed sparsity vector v^i. We want to use at most $O(\log n)$ compressed sparsity vectors $C(\hat{r}^{j_0,y_0}), C(\hat{r}^{j_1,y_1}), C(\hat{r}^{j_2,y_2}), C(\hat{r}^{j_3,y_3}), \ldots$ to compute v^i. We choose the pairs j_p, y_p according to the binary expansion of i: we choose different integers j_p such that $i = 2^{j_0}+2^{j_1}+2^{j_2}+2^{j_3}+\ldots$. We also choose the groups y_p so that different groups do not share nodes and together the groups cover all elements $1, 2, \ldots, i$ from the spine. To compute v^i, we could compute the $(\max,+)$-convolution of the vectors $\hat{r}^{j_0,y_0}, \hat{r}^{j_1,y_1}, \hat{r}^{j_2,y_2}, \ldots$ from which we can obtain the compressed vector v^i. However, this would take too much time. Instead, we observed in our previous attempt that we can compute v^i directly from the compressed vectors $C(r^{j_p,y_p})$. Thus, the compressed sparsity vector v^i can be computed in time $O(\log n \cdot \log^2(nx_{\max})/\delta'^2)$.

Now we have l compressed sparsity vectors $v^1, v^2, \ldots, v^l$. To compute the sparsity vector for the tree x we could do the following: get sparsity vectors corresponding to vectors $v^1, v^2, \ldots, v^l$ and compute entry-wise minimum of the l vectors. This has too large time complexity and instead we compute the answer by computing the entry-wise minimum without explicitly computing the sparsity vectors. More precisely, let $v' \in \mathbb{N}^n$ be a vector consisting of only $+\infty$ initially. For every $i \in [l]$ and for every entry v^i_j, we set $v'_{v^i_j}$ to be equal to $\min(v'_{v^i_j}, j)$. We set $z = 1$. For every $j = 1, \ldots, n$ in this order, we set $v_j := (1+\delta')^z$ and, if $v'_j \neq +\infty$, we update $z = \max(z, v'_j)$. We output v as the sparsity vector for the tree x.

The approximation factor of the algorithm We assume that the vectors s^i (that we compute recursively) are $(1+\delta)$-approximations to the exact sparsity vectors. We compute the vectors $\hat{r}^{j,y}$ that introduce another $1+\delta'$ multiplicative error in the approximation. We then get compressed sparsity vectors $C(\hat{r}^{j,y})$, which gives another multiplicative error factor $1+\delta'$. To compute every v^i, we need to compute a $(\max,+)$-convolution between compressed sparsity vectors $O(\log n)$ times, which gives $(1+\delta')^{O(\log n)}$ error. The total error from the recursive call is bounded by

$$(1+\delta)\cdot(1+\delta')\cdot(1+\delta')\cdot(1+\delta')^{O(\log n)}$$
$$= (1+\delta)\cdot(1+\delta')^{O(\log n)} .$$

Since the depth of the recursion is $O(\log n)$ (from the definition of the heavy-light decomposition), the error of the algorithm is $(1+\delta')^{O(\log^2 n)}$. To make it smaller than $1+\varepsilon$, we set $\delta' = \Theta(\varepsilon/\log^2 n)$.

The runtime of the algorithm The runtime of the recursive step is dominated by computing the vectors $\hat{r}^{j,y}$. Plugging $\delta' = \Theta(\varepsilon/\log^2 n)$ into (3), we get that the runtime of the recursive step is $O(\frac{n}{\varepsilon^2}\cdot\log^{11} n \cdot \log^2 x_{\max})$. Since the depth of the recursion of the algorithm is $O(\log n)$, the final runtime is $O(\frac{n}{\varepsilon^2}\cdot\log^{12} n\cdot\log^2 x_{\max})$.

Acknowledgments

This work was supported in part by an IBM PhD Fellowship, a Google PhD Fellowship, the NSF, MADALGO, and the Simons Foundation. Some of the work was performed while Ludwig Schmidt was a visiting student at UC Berkeley and then an intern in the Google Brain team.

We would like to thank Amir Abboud for very helpful comments regarding the complexity of $(\max,+)$-convolution.

References

[Abb16] Amir Abboud. personal communication, 2016.

[AGW15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1681–1697. SIAM, 2015.

[AVWY15] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 41–50. ACM, 2015.

[AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 434–443. IEEE, 2014.

[BB94] Marko Bohanec and Ivan Bratko. Trading accuracy for simplicity in decision trees. *Machine Learning*, 15(3):223–250, 1994.

[BCD$^+$06] David Bremner, Timothy M Chan, Erik D Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and x+ y. In *Algorithms–ESA 2006*, pages 160–171. Springer, 2006.

[BCDH10] Richard G. Baraniuk, Volkan Cevher, Marco F. Duarte, and Chinmay Hegde. Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4):1982–2001, 2010.

[BDT16] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight Hardness Results for Maximum Weight Rectangles. In *International Colloquium on Automata, Languages, and Programming*, 2016.

[BJ93] Richard G. Baraniuk and Douglas L. Jones. A signal-dependent time-frequency representation: optimal kernel design. *IEEE Transactions on Signal Processing*, 41(4):1589–1602, 1993.

[CL15] Timothy M Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 31–40. ACM, 2015.

[CT13] Coralia Cartis and Andrew Thompson. An exact tree projection algorithm for wavelets. *IEEE Signal Processing Letters*, 20(11):1026–1029, 2013.

[Don97] David L. Donoho. Cart and best-ortho-basis: a connection. *Annals of Statistics*, 25(5):1870–1911, 1997.

[HIS14a] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. A fast approximation algorithm for tree-sparse recovery. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1842–1846, 2014.

[HIS14b] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Nearly linear-time model-based compressive sensing. In *Automata, Languages, and Programming (ICALP)*, volume 8572 of *Lecture Notes in Computer Science*, pages 588–599. 2014.

[HIS15a] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Approximation algorithms for model-based compressive sensing. *IEEE Transactions on Information Theory*, 61(9):5129–5147, 2015. Conference version appeard in the *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2014).

[HIS15b] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Fast algorithms for structured sparsity. *Bulletin of EATCS*, 3(117), 2015.

[HIS15c] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. A nearly-linear time framework for graph-structured sparsity. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 928–937. JMLR Workshop and Conference Proceedings, 2015.

[LZ93] D Lozovanu and A Zelikovsky. Minimal and bounded tree problems. *Tezele Congresului XVIII al Academiei Romano-Americane*, pages 25–26, 1993.

[NvLvdZ12] Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. Reducing a target interval to a few exact queries. In *International Symposium on Mathematical Foundations of Computer Science*, pages 718–727. Springer, 2012.

[PS16] Julianus Pfeuffer and Oliver Serang. A bounded p-norm approximation of max-convolution for sub-quadratic bayesian inference on additive factors. *Journal of Machine Learning Research*, 17(36):1–39, 2016.

[RSM$^+$96] Ramamurthy Ravi, Ravi Sundaram, Madhav V Marathe, Daniel J Rosenkrantz, and Sekharipuram S Ravi. Spanning trees-short or small. *SIAM Journal on Discrete Mathematics*, 9(2):178–200, 1996.

[RZ04] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *Algorithms–ESA 2004*, pages 580–591. Springer, 2004.

[Ser15] Oliver Serang. Fast computation on semirings isomorphic to $(\times, \max)$ on $\mathbb{R}_+$. *arXiv preprint arXiv:1511.05690*, 2015.

[ST83] Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654. IEEE, 2010.

[WW13] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.

[Zwi98] Uri Zwick. All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 310–319. IEEE, 1998.

A $(\max,+)$-Convolution hardness for the tree sparsity problem

In this section is we provide an evidence that the exact tree sparsity requires nearly quadratic time. Recall that, given a binary node-weighted tree x of size m, we want to output the largest sum of weights of nodes of x that we can pick up by choosing a rooted subtree of x of size $k \le m$. The best known algorithm for this problem runs in time $\Theta(m^2)$ (see Section D). In this section we show that this problem cannot be solved in a strongly subquadratic time unless the $(\max,+)$-convolution problem can be solved in a strongly subquadratic time. This is a well studied problem [BCD$^+$06, CL15] which is known to be at least as hard as the *polyhedral 3SUM* problem[6] [BCD$^+$06].

Theorem 6. *Let x be a binary node-weighted tree of size m. If $(\max,+)$convolution cannot be computed in $m^{2-\Omega(1)}$ time, then the sparsity cannot be computed in $m^{2-\Omega(1)}$ time either.*

Proof. Follows from Theorems 10 and 14 below. □

Definition 7 (SUM_1 problem). *Given three vectors $A, B, C \in \mathbb{Z}^n$, output*

$$\{k \mid \exists i,j \ : \ k = i+j \text{ and } A_i + B_j + C_k \ge 0\}.$$

Definition 8 (SUM_2 problem). *Given three vectors $A, B, C \in \mathbb{Z}^n$, output $t = 0, \dots, n-1$ such that*

$$t \in \{k \mid \exists i,j \ : \ k = i+j \text{ and } A_i + B_j + C_k \ge 0\}$$

or report that there is no such an integer.

Definition 9 (SUM_3 problem). *Given three vectors $A, B, C \in \mathbb{Z}^n$, decide if the following statement is true:*

$$\exists i,j \ : \ A_i + B_j + C_{i+j} \ge 0.$$

If $(\max,+)$-Convolution problem can be solved in strongly subquadratic time, so can SUM_3 problem. We will show the opposite direction - if SUM_3 problem can be solved in strongly subquadratic time, then $(\max,+)$-Convolution problem can be solved in strongly subquadratic time. We show this by reducing $(\max,+)$-Convolution problem to SUM_1 problem, SUM_1 problem to SUM_2 problem, SUM_2 problem to SUM_3 problem. Our proof of the following theorem uses the approach from [WW10] (see Sections 4.2 and 8).

Theorem 10. *If SUM_3 problem can be solved in a strongly subquadratic time, then $(\max,+)$-Convolution can be solved in a strongly subquadratic time.*

Proof. Follows from Lemmas 11, 12 and 13 below. □

It can be shown that SUM_3 problem is no harder than the $3SUM$ problem (given a set of integers, decide if it contains three integers that sum up to 0) [Abb16]. This can be done using the techniques from [NvLvdZ12] and [WW13] (see Theorem 3.3). This together with Theorem 10 implies that $(\max,+)$-convolution problem is no harder than the $3SUM$ problem (up to a factor that is logarithmic in the largest absolute value of an integer in the input).

[6] Given three vectors $A = (A_0, \dots, A_{n-1})^T$, $B = (B_0, \dots, B_{n-1})^T$, and $C = (C_0, \dots, C_{n-1})^T$, such that $A_i + B_j \le C_{i+j}$ for all $0 \le i,j < n$, decide whether $A_i + B_j = C_{i+j}$ for any $0 \le i,j < n$. No strongly subquadratic time algorithm is known for this problem.

Lemma 11. *Let $A, B \in \mathbb{Z}^n$ be input for* $(\max, +)$*-Convolution problem. Let W be the largest absolute value of an integer appearing in vector A or B. If SUM_1 can be solved in time $O(n^{2-\varepsilon})$, then* $(\max, +)$*-Convolution can be solved in $O(n^{2-\varepsilon} \cdot \log W)$ time.*

Proof. To solve $(\max, +)$-Convolution, we perform $O(\log W)$ steps of binary search for all indices in parallel.

We define two vectors $L_k = -10W$ and $H_k = 10W$ for all $k = 0, \dots, n-1$. We perform the following sequence of steps $100 \cdot \log W$ times:

1. Set $A' \leftarrow A$, $B' \leftarrow B$ and $C'_k \leftarrow -\left\lfloor \frac{L_k + H_k}{2} \right\rfloor$ for all $k = 0, \dots, n-1$.
2. Run the algorithm for SUM_1 on vectors A', B' and C'. Let R be the output set.
3. For every $k = 0, \dots, n-1$, consider two cases. If $k \in R$, then update $L_k \leftarrow \left\lfloor \frac{L_k + H_k}{2} \right\rfloor$. If $k \notin R$, then update $H_k \leftarrow \left\lfloor \frac{L_k + H_k}{2} \right\rfloor$.

Let C be the output of $(\max, +)$-Convolution instance that we need to output. During the execution of the algorithm above, we always maintain property that $L_k \leq C_k < H_k$ for all $k = 0, \dots, n-1$. After $100 \cdot \log W$ iterations, we have $L_k = H_k - 1$. This implies that $L_k = C_k$ for all $k = 0, \dots, n-1$. Therefore, we output L_k as the answer. $\square$

Lemma 12. *If SUM_2 can be solved in time $O(n^{2-\varepsilon})$, then SUM_1 can be solved in time $O(n^{2-(\varepsilon/2)})$.*

Proof. The idea is to run the algorithm for SUM_2 problem to find element t of interest, remove C_t from vector C and repeat until we found all elements that need to be reported. To achieve the stated runtime, we split both vectors A and B in $\sqrt{n}$ vectors each having $\sqrt{n}$ entries. Then we run the algorithm for SUM_2 for each pair of shorter vectors at least once. Below we provide more details.

Let $A = (A_0, \dots, A_{n-1})^T$, $B = (B_0, \dots, B_{n-1})^T$ and $C = (C_0, \dots, C_{n-1})^T$ be the input vectors for SUM_1 problem. Let W be the largest absolute value of an integer in vector A or B. Let $T = \emptyset$ be the set that the algorithm will output. Initially the set is empty set. Let a be parameter that we will set later (we will set $a = 1/2$). For each pair of integer (i', j'), $i' = 0, \dots, n^{1-a} - 1$, $j' = 0, \dots, n^{1-a} - 1$, we perform the following sequence of steps (in total, we do the sequence of steps n^{2-2a} times).

1. For $i = 0, \dots, n^a - 1$, we set $A'_i = A_{(i' \cdot n^a) + i}$. For $i = n^a, \dots, 2n^a - 2$, set $A'_i = -10W$.
2. For $j = 0, \dots, n^a - 1$, we set $B'_j = A_{(j' \cdot n^a) + j}$. For $j = n^a, \dots, 2n^a - 2$, set $B'_j = -10W$.
3. For $k = 0, \dots, 2n^a - 2$, we set $C'_k = C_{((i' + j') \cdot n^a) + k}$.
4. Run the algorithm for SUM_2 problem on vectors A', B' and C'. If the algorithm outputs an integer t, add integer t to set T and set $C_t = -10W$, and go to Step 1.

The correctness of the algorithm follows from its description. It remains to analyse its runtime. Suppose that the algorithm for SUM_2 runs in time $O(n^{2-\varepsilon})$. Since the length of vectors A', B' and C' is $O(n^a)$, every invocation of SUM_2 algorithm takes $S_1 := O(n^{a(2-\varepsilon)})$ time. We run algorithm for SUM_2 for every tuple (i', j'). The number of tuples is $S_2 := O(n^{2-2a})$. In Step 4 we might need to reiterate execution of the sequence of steps if we received some integer t. However, notice that no integer t can be reported twice by the definition of SUM_2 problem and because we set $C_t = -10W$. Therefore, the number of times we might need to reiterate the sequence of steps, is upper bounded by $S_3 := n$. Thus, the total runtime is upper bounded by

$$O(S_1 \cdot S_2 + S_1 \cdot S_3) = O\left(n^{a(2-\varepsilon)} \cdot n^{2-2a} + n^{a(2-\varepsilon)} \cdot n\right).$$

By setting $a = 1/2$, we get that the runtime is upper bounded by $O(n^{2-(\varepsilon/2)})$, as required. □

Lemma 13. *If SUM_3 problem can be solved in time $O(n^{2-\varepsilon})$, then SUM_2 problem can be solved in time $O(n^{2-\varepsilon} \cdot \log n)$.*

Proof. This follows by binary search. □

Theorem 14. *If tree sparsity can be solved in strongly subquadratic time, then SUM_3 problem can be solved in a strongly subquadratic time as well.*

Proof. Let $A, B, C \in \mathbb{Z}^n$ be the input to the SUM_3 problem. Let W be equal to 10 times the largest absolute value of an entry in A, B, C. Consider $A \in \mathbb{Z}^n$. We first construct a path p_A of n nodes such that the weight of the first node is $11W + A_0$ and the weight of the i-th nodes is $W + A_{i-1} - A_{i-2}$ for $i = 2, \ldots, n$. p_B is constructed in the same way. Finally, we construct a path p_C such that the weight of the first node is $11W + C_{n-1}$ and the weight of the i-th nodes is $W + C_{n-i} - C_{n-i+1}$ for $i = 2, \ldots, n$. We then build the tree x as follows. The root of the tree is equal to the first node of p_A. The left child of the root is equal to the second node of p_A. The right child of the root is equal to the first node of p_B. (call it v). The left child of v is equal to the second node of p_B. Finally, the right child of v is equal to the first node of p_C. We set m to be the size of the tree (which is $3n$), and set k to be equal to $n + 2$. Observe that

$$\max_{\Omega \in \mathbb{T}_k} \|x_\Omega\|_1 \geq (2 + n)W + 30W$$

iff there are $i, j \in \{0, 1, \ldots, n-1\}$ such that $A_i + B_j + C_{i+j} \geq 0$.

To show the above inequality, let $k_A, k_B, k_C \geq 1$ be the number of nodes that we pick up from paths p_A, p_B, p_C, respectively, in the tree x in the optimal support Ω. k_A, k_B, k_C are positive because we assign very large weights to the first nodes of the paths. We have that $k_A + k_B + k_C = k = 2 + n$. It is easy to verify that the contributions from paths p_A, p_B, p_C are $10W + k_A W + A_{k_A - 1}, 10W + k_B W + B_{k_B - 1}, 10W + k_C W + C_{n - k_C}$, respectively. Since $k_A + k_B + k_C = 2 + n$, the total contribution from the three paths is $30W + (2 + n)W + A_{k_A - 1} + B_{k_B - 1} + C_{(k_A - 1) + (k_B - 1)}$, as required. □

B Constant factor head approximation in nearly linear time

Before we give the main algorithm of this section, we setup some auxiliary sub-routines.

B.1 Subroutines

First, we state the following sub-routine which extracts a subtree of bounded size and proportional "density" from a given tree. A similar sub-routine has appeared before in [HIS15c], but here we give a variant that maintains exact sparsity.

Algorithm 1 Extracting a dense subtree.

```
1: function DenseSubtree($x, \Omega, k'$)
2:     Let $T$ be a tour through the nodes in $\Omega$ as they appear in a depth-first traversal.
3:     Let $I = (i_1, \ldots, i_{2|\Omega|-1})$ be the node indices in order of the tour $T$.
4:     Let $x'_j = \begin{cases} x_{i_j} & \text{if position } j \text{ is the first appearance of } i_j \text{ in } I \\ 0 & \text{otherwise} \end{cases}$
5:     Let $S = (i_j, \ldots, i_{j+k'-1})$ be a contiguous subsequence of $I$ with $\frac{1}{k'} \sum_{\ell=j}^{j+k'-1} x'_\ell \geq \frac{1}{2|\Omega|} \|x_\Omega\|_1$.
6:     return $\Omega'$, the set of nodes in $S$.
```

Lemma 15. *Let $x \in \mathbb{R}^n_+$ be a vector of node weights and let Ω be a subtree. Moreover, let $k' \in \mathbb{N}$ be the target sparsity. Then* DenseSubtree(x, Ω, k') *returns a subtree Ω' of size $|\Omega'| \leq k'$ such that*

$$\|x_{\Omega'}\|_1 \;\geq\; \frac{k'}{2|\Omega|}\|x_\Omega\|_1 \;.$$

Moreover, DenseSubtree *runs in time $O(|\Omega|)$.*

Proof. We use the notation set up in the algorithm DenseSubtree (see Algorithm 1). By a simple averaging argument, we know that at least one contiguous length-k' subsequence of I achieves at least the density of the sequence I, which is

$$\frac{1}{|I|}\sum_{j=1}^{|I|} x'_j \;>\; \frac{1}{2|\Omega|}\sum_{j=1}^{|I|} x'_j \;=\; \frac{1}{2|\Omega|}\|x_\Omega\|_1 \;.$$

The second inequality follows from the definition of x' (every node value in Ω appears exactly once in the sum).

Therefore, Line 5 of the algorithm always succeeds and we find a subset $\Omega' \subseteq \Omega$ corresponding to a subsequence $S = (i_j, \ldots, i_{j+k'-1})$ such that

$$\|x_{\Omega'}\|_1 \;=\; \sum_{\ell=j}^{j+k'-1} x'_\ell \;\geq\; \frac{k'}{2|\Omega|}\|x_\Omega\|_1 \;.$$

Finally, we can find such a dense contiguous sequence in linear time by maintaining a sliding window over the sequence I. □

Moreover, we use the following sub-routine for solving the Lagrangian relaxation of the tree sparsity problem [HIS14a].

Fact 16 ([HIS14a])**.** *There is an algorithm* SolveRelaxation *with the following guarantee. Let $x \in \mathbb{R}^n_+$ be a vector of node weights and let λ be the Lagrangian trade-off parameter. Then* SolveRelaxation(x, λ) *returns a subtree Ω such that*

$$\|x_\Omega\|_1 - \lambda|\Omega| \;\geq\; \max_{\Omega' \in \mathbb{T}} \|x_{\Omega'}\|_1 - \lambda|\Omega'| \;. \tag{4}$$

Moreover, SolveRelaxation *runs in time $O(n)$.*

Algorithm 2 Finding an unrooted head approximation.

```
1: function UnrootedHead($x, k, \alpha$)
2:     $x_{\max} \leftarrow \max_{i \in [n]} x_i$
3:     $\lambda_l \leftarrow \frac{x_{\max}}{2k}$
4:     $\Omega_l \leftarrow$ SolveRelaxation($x, \lambda_l$)
5:     if $|\Omega_l| \leq 2k$ then
6:         return DenseSubtree($x, \Omega_l, \alpha k$)
7:     $\lambda_r \leftarrow 2\|x\|_1$
8:     $\varepsilon \leftarrow \frac{x_{\max}}{4k}$
9:     while $\lambda_r - \lambda_l > \varepsilon$ do
10:        $\lambda_m \leftarrow \frac{\lambda_l + \lambda_r}{2}$
11:        $\Omega_m \leftarrow$ SolveRelaxation($x, \lambda_m$)
12:        if $|\Omega_m| > 2k$ then
13:            $\lambda_l \leftarrow \lambda_m$
14:        else
15:            $\lambda_r \leftarrow \lambda_m$
16:    $\Omega_l \leftarrow$ SolveRelaxation($x, \lambda_l$)
17:    $\Omega'_l \leftarrow$ DenseSubtree($x, \Omega_l, \alpha k$)
18:    $\Omega_r \leftarrow$ SolveRelaxation($x, \lambda_r$)
19:    $\Omega'_r \leftarrow$ DenseSubtree($x, \Omega_r, \alpha k$)
20:    if $\|\Omega'_l\|_1 \geq \|\Omega'_r\|_1$ then
21:        return $\Omega'_l$
22:    else
23:        return $\Omega'_r$
```

B.2 Unrooted head approximation

We now give an algorithm for finding an *unrooted* subtree that achieves a constant-factor head approximation. While the resulting subtree is not connected to the root, we have sufficiently tight control over the sparsity so that we can later post-process the subtree by connecting it to the root.

Theorem 17. *Let $x \in \mathbb{R}^n_+$ be a vector of node weights, let $k \in \mathbb{N}$ be the target sparsity, and let $0 < \alpha < 1$ be a sparsity control parameter. Then* UnrootedHead(x, k, α) *returns a subtree Ω of size $|\Omega| \leq \alpha k$ such that*

$$\|x_\Omega\|_1 \;\geq\; \frac{\alpha}{16} \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1 \,.$$

Moreover, UnrootedHead *runs in time $O(n \log n)$.*

Proof. We adopt the notation of Algorithm 2. Moreover, let Ω^* be an optimal subtree of size k, i.e., we have $\Omega^* \in \mathbb{T}_k$ and $\|x_{\Omega^*}\|_1 = \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1$.

There are three cases in which UnrootedHead returns a subtree: Lines 6, 21, and 23. We consider these three cases separately. Note that in every case, the final subtree is the result of a call to DenseSubtree with sparsity parameter αk. Hence the final subtree Ω returned by UnrootedHead always satisfies $|\Omega| \leq \alpha k$ (see Lemma 15). It remains to show that the subtree Ω also satisfies the desired head approximation guarantee.

Case 1: We start with Line 6. Substituting Ω^* into the guarantee of SolveRelaxation (see Fact 16), we get the following inequalities:

$$\begin{aligned} \|x_{\Omega_l}\|_1 - \lambda_l|\Omega_l| \;&\geq\; \max_{\Omega' \in \mathbb{T}} \|x_{\Omega'}\|_1 - \lambda_l|\Omega'| \;\geq\; \|x_{\Omega^*}\|_1 - \lambda_l k \\ \|x_{\Omega_l}\|_1 \;&\geq\; \|x_{\Omega^*}\|_1 - \lambda_l(k - |\Omega_l|)\;. \end{aligned} \tag{5}$$

Equation (5) and its variant for λ_r will become useful again later in the proof. Now, we substitute $\lambda_l = \frac{x_{\max}}{2k}$ and get

$$\|x_{\Omega_l}\|_1 \;\geq\; \|x_{\Omega^*}\|_1 - \frac{1}{2}x_{\max}\;.$$

Since we can assume that the sparsity k is at least the depth of the tree, a k-sparse rooted subtree can always pick up the largest node weight $x_{\max}$. So $\|x_{\Omega^*}\|_1 \geq x_{\max}$ and hence $\|x_{\Omega_l}\|_1 \geq \frac{1}{2}\|x_{\Omega^*}\|_1$.

To complete this case, we invoke the guarantee of DenseSubtree (Lemma 15) to get

$$\begin{aligned} \|x_\Omega\|_1 \;&\geq\; \frac{\alpha k}{2|\Omega_l|}\|x_{\Omega_l}\|_1 \\ &\geq\; \frac{\alpha}{4}\|x_{\Omega_l}\|_1 \\ &\geq\; \frac{\alpha}{8}\|x_{\Omega^*}\|_1\;. \end{aligned}$$

Case 2: We now consider the case that the algorithm performs the binary search over λ and returns in Line 21 or 23. Note that we initialize the binary search so that we always have $|\Omega_l| > 2k$ and $|\Omega_r| \leq 2k$. Moreover, at the end of the binary search we also have $\lambda_l \leq \lambda_r \leq \lambda_l + \varepsilon$.

We now distinguish two sub-cases: in the first sub-case, we assume that the solution $|\Omega_l|$ has a good density $\frac{\|x_{\Omega_l}\|_1}{|\Omega_l|}$, which implies that the subtree Ω'_l is sufficiently good. In the complementary case, we can then use the low density of Ω_l to show that Ω'_r is a good solution.

Sub-case 2a: $\frac{\|x_{\Omega_l}\|_1}{|\Omega_l|} \;\geq\; \frac{1}{4}\frac{\|x_{\Omega^*}\|_1}{k}$. Substituting this inequality into the guarantee provided by DenseSubtree gives

$$\|x_{\Omega'_l}\|_1 \;\geq\; \frac{\alpha k}{2|\Omega_l|}\|x_{\Omega_l}\|_1 \;\geq\; \frac{\alpha}{8}\|x_{\Omega^*}\|_1$$

as desired.

Sub-case 2b: $\frac{\|x_{\Omega_l}\|_1}{|\Omega_l|} < \frac{1}{4}\frac{\|x_{\Omega^*}\|_1}{k}$. We lower bound $\|x_{\Omega_r}\|_1$ via the guarantee provided by SolveRelaxation, which we re-arrange as in Case 1 to get:

$$\begin{aligned} \|x_{\Omega_r}\|_1 \;&\geq\; \|x_{\Omega^*}\|_1 - \lambda_r(k - |\Omega_r|) \\ &\geq\; \|x_{\Omega^*}\|_1 - \lambda_r k\;. \end{aligned} \tag{6}$$

In order to control the RHS above, we need an upper bound on λ_r (note that $|\Omega_r|$ can be less than k). We establish this via an upper bound on λ_l and using that λ_l and λ_r are close at the end of the binary search. Re-arranging Equation (5) gives:

$$\begin{aligned}\lambda_l &\leq \frac{\|x_{\Omega_l}\|_1 - \|x_{\Omega^*}\|_1}{|\Omega_l| - k} \\ &\leq \frac{\|x_{\Omega_l}\|_1}{|\Omega_l| - k} \\ &< \frac{2\|x_{\Omega_l}\|_1}{|\Omega_l|} \\ &\leq \frac{1}{2}\frac{\|x_{\Omega^*}\|_1}{k}\end{aligned}$$

where we used $|\Omega_l| > 2k$ and the low-density assumption for Ω_l in this sub-case.

Substituting this upper bound and $\lambda_r \leq \lambda_l + \varepsilon$ back into Equation (6) gives

$$\begin{aligned}\|x_{\Omega_r}\|_1 &\geq \|x_{\Omega^*}\|_1 - \frac{1}{2}\|x_{\Omega^*}\|_1 - \varepsilon k \\ &\geq \frac{1}{4}\|x_{\Omega^*}\|_1\end{aligned}$$

where we used $x_{\max} \leq \|x_{\Omega^*}\|_1$ as in Case 1.

We now invoke the guarantee of DenseSubtree. As mentioned above, we maintain $|\Omega_r| \leq 2k$ as an invariant in the binary search. Hence we get

$$\|x_{\Omega'_r}\|_1 \geq \frac{\alpha k}{2|\Omega_r|}\|x_{\Omega_r}\|_1 \geq \frac{\alpha}{16}\|x_{\Omega^*}\|_1 .$$

Finally, we prove the running time of UnrootedHead. Since both subroutines SolveRelaxation and DenseSubtree run in linear time, the overall time complexity is dominated by the binary search. We can upper bound the number of iterations by the logarithm of

$$\frac{\lambda_r}{\varepsilon} = \frac{8k\|x\|_1}{x_{\max}} \leq 8nk \leq 8n^2 ,$$

which implies the running time bound in the theorem. □

B.3 Final head approximation algorithm

We now state our overall head approximation algorithm. The main idea is to invoke UnrootedHead form the previous subsection with a sufficiently small sparsity control parameter α so that we can connect the resulting subtree to the root without violating our sparsity constraint.

Theorem 18. *There is an algorithm* HeadApprox *with the following guarantee. Let $x \in \mathbb{R}^n_+$ be a vector of node weights, let d be the depth of the tree, and let $k \in \mathbb{N}$ be the target sparsity. Then* HeadApprox *returns a rooted subtree Ω of size $|\Omega| \leq k$ such that*

$$\|x_\Omega\|_1 \geq \frac{k-d}{16k}\max_{\Omega' \in \mathbb{T}_k}\|x_{\Omega'}\|_1 .$$

Moreover, HeadApprox *runs in time $O(n \log n)$.*

Proof. Let Ω' be the subtree returned by UNROOTEDHEAD$(x, k, \frac{k-d}{k})$. Hence Ω' satisfies $|\Omega'| \leq k-d$ and

$$\|x_{\Omega'}\|_1 \ \geq \ \frac{k-d}{16k}$$

Next, let Ω_p be the path from the root of the subtree Ω' to the root of the overall tree. Since the depth of the overall tree is d, we have $|\Omega_p| \leq d$.

We now let $\Omega = \Omega' \cup \Omega_p$ be the final subtree. Clearly, Ω is a rooted subtree and still satisfies the same head approximation guarantee as Ω'. Moreover, we have $|\Omega| = |\Omega'| + |\Omega_p| \leq d$ as desired.

The running time of HEADAPPROX follows directly from Theorem 17. □

Corollary 19. *Assume that the depth of the input tree is at most a constant fraction of the sparsity k. Then* HEADAPPROX *is a constant-factor head-approximation algorithm with exact sparsity k.*

B.4 A boosted algorithm

Prior work shows that it is possible to "boost" a head-approximation algorithm so that the approximation ratio improves arbitrarily close to one while only incurring a small increase in sparsity [HIS15a].

Fact 20 ([HIS15a]). *Let* HEAD *be a head-approximation algorithm with approximation ratio c_H and output sparsity αk. Then* BOOSTEDHEAD$_t$ *is a head-approximation algorithm with approximation ratio $c'_H = 1 - (1 - c_H)^t$ and output sparsity $t\alpha k$. Moreover,* BOOSTEDHEAD *runs in time $O(t \cdot T_{\text{HEAD}})$, where T_{HEAD} is the time complexity of a single invocation of the algorithm* HEAD.

We can invoke Fact 20 and our new head approximation algorithm to give a range of trade-offs between sparsity increase and head-approximation ratio. It is worth noting that Fact 20 has only a logarithmic dependence on the gap between c_H and an exact head "approximation" with factor 1. For node weights x coming from a bounded range, this allows us to achieve an exact head (and hence also tail) approximation with only logarithmic increase in sparsity. More precisely, we get the following theorem, where we assume that the depth of the tree d is at most $k/2$ in order to simplify the bounds.

Theorem 21. *There is an algorithm* BOOSTEDHEADAPPROX *with the following guarantee. Let $x \in \mathbb{N}_+^n$ be a vector of node weights bounded as $x_i \leq \Delta$, and let $k \in \mathbb{N}$ be the target sparsity. Then* BOOSTEDHEADAPPROX(x, k) *returns a rooted subtree Ω such that $|\Omega| \leq 33k \log 2k\Delta$ and*

$$\|x_\Omega\|_1 \ \geq \ \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1 \, .$$

Moreover, BOOSTEDHEADAPPROX *runs in time $O(n\,(\log n)\,(\log k\Delta))$.*

Proof. First, note that we can restrict our attention to trees with depth at most k. Any node that has larger distance from the root cannot participate in the optimal solution due to the sparsity constraint.

Next, we show that increasing our output sparsity by a constant factor allows us to get a result independent of the depth of the tree (this is in contrast to Theorem 18 and Corollary 19). We can construct an algorithm HEADAPPROX' that achieves a head approximation ratio of $\frac{1}{32}$ by invoking UNROOTEDHEAD from Theorem 17 with parameter $\alpha = \frac{1}{2}$. We ensure that the output is an unrooted tree by connecting it to the root. This increases the sparsity by at most the depth, which we just bounded by k. Hence the total sparsity of the output is $\alpha k + k = \frac{3}{2}\alpha$. Moreover, HeadApprox' runs in time $O(n \log n)$.

We now boost our new head approximation algorithm HeadApprox'. Rearranging the guarantee in Fact 20 shows that

$$t \;=\; \frac{\log \frac{1}{\varepsilon}}{\log \frac{1}{1-c_H}} \;\leq\; 22 \log \frac{1}{\varepsilon}$$

suffices for a boosted head approximation ratio of $c'_H = 1 - \varepsilon$.

Since we have integer node weights, the gap between an optimal head approximation and the second-best possible head approximation is at least 1. Moreover, the total weight of an optimal k-sparse subtree is at most $k\Delta$. Hence it suffices to set $\varepsilon = \frac{1}{2k\Delta}$ in order to guarantee that a $(1-\varepsilon)$-head approximation is exact. As a result, invoking BoostedHead with $t = 22 \log 2k\Delta$ produces a subtree Ω with the desired properties.

Each iteration of BoostedHead invokes HeadApprox' once. So the running time of HeadApprox' (see Theorem 18) and our bound on t imply the running time bound of the theorem. □

C Approximating $(\max, +)$ and $(\min, +)$-convolutions

C.1 Approximating $(\max, +)$ and $(\min, +)$-convolutions between two sequences

Let $A, B \in \mathbb{N}^n$ be two integer vectors with positive entries. Let W be the largest value of an entry in A or B. Let $C \in \mathbb{N}^n$ be the $(\max, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1+\varepsilon)C'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W\right).$$

An analogous statement holds if C is the $(\min, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1+\varepsilon)C'_k$ for all $k = 0, \ldots, n-1$ in the same runtime.

Proof. Given a vector $D \in \mathbb{N}^n$ with positive entries and an integer $i \geq 0$, we define a binary vector $\chi(D, i)$:

$$\chi(D, i)_k := \begin{cases} 1 & \text{if } (1+\varepsilon)^i \leq D_k < (1+\varepsilon)^{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

For all pairs of integers $0 \leq i, j \leq \log_{1+\varepsilon} W$, define vector $\chi_{i,j} := \chi(A, i) * \chi(B, j)$. Computing all vectors $\chi_{i,j}$ takes total time

$$O\left((\log^2_{1+\varepsilon} W) \cdot n \log n\right).$$

Finally, we set

$$C'_k := \max_{\substack{0 \leq i,j \leq \log_{1+\varepsilon} W \\ \text{s.t. } (\chi_{i,j})_k = 1}} (1+\varepsilon)^i + (1+\varepsilon)^j \tag{7}$$

for all $k = 0, \ldots, n-1$. The runtime and the correctness follows from the description.

The proof for $(\min, +)$ convolution is analogous. □

C.2 Approximating $(\max,+)$ and $(\min,+)$-convolutions between multiple sequences

Let $A^1 \in \mathbb{N}^{n_1}, \ldots, A^l \in \mathbb{N}^{n_l}$ be $l \geq 2$ vectors with positive entries. We assume that the entries of the vectors are indexed starting with 0, i.e., $A^i = \left(A^i_0, \ldots, A^i_{n_i-1}\right)^T$ for every $i = 1, \ldots, l$. Let $B \in \mathbb{N}^n$, $n = n_1 + \ldots + n_l$ be the $(\min,+)$-convolution between the l vectors $A^1, \ldots, A^{n_l}$ defined as follows:

$$B_j = \min_{\substack{m_1+\ldots+m_l=j \\ \text{s.t. } 0\leq m_t \leq n_t-1 \ \forall t=1,\ldots,l}} \left(A^1_{m_1} + \ldots + A^l_{m_l}\right)$$

for every $j = 0, \ldots, n-1$. We can output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W \cdot \log^3 l\right),$$

where W denotes the largest entry in the vectors $A^1, \ldots, A^l$.

An analogous statement holds for $(\max,+)$-convolution (we replace min in the equation for B_j with max). The runtime stays the same and we output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$.

Proof. We repeatedly use the fast algorithm for approximately computing $(\min,+)$-convolution. Let B'' be the $(\min,+)$-convolution between the vectors $A_1, \ldots, A_{\lfloor l/2 \rfloor}$ and let B''' be the $(\min,+)$convolution between the vectors $A_{\lfloor l/2 \rfloor+1}, \ldots, A_l$. Then B is the $(\min,+)$convolution between the *two* vectors B'' and B'''. This gives a natural recursive algorithm for computing B': recursively approximate B'' and B''' and use the approximation algorithm from Theorem 2.2 to compute get B'. If B'' and B''' are approximated within factor $(1+\varepsilon')$ and we set approximation factor to be $(1+\varepsilon'')$ in the algorithm from Theorem 2.2, we get $(1+\varepsilon')(1+\varepsilon'')$ approximation factor for B'. Since the depth of the recursion is $\log_2 l$, we get that the final approximation factor of B is $(1+\varepsilon'')^{\log_2 l}$. Setting $\varepsilon'' := O\left(\frac{\varepsilon}{\log l}\right)$ gives the required approximation factor for B'. Since the number of the recursion levels is $O(\log l)$ and every level takes $O\left(\frac{n}{\varepsilon''^2} \cdot \log n \cdot \log^2 W\right)$, we get the required runtime.

The proof for $(\max,+)$ is analogous. □

D Tree Sparsity in time $O(kn)$ on unbalanced trees

Theorem 22. *Given a binary tree of size n and an integer k, we can solve the Tree Sparsity problem in time $O(n^2)$.*

Proof. We show this inductively.

Consider the recursive algorithm for solving the Tree Sparsity problem. Consider a tree with the left subtree is of size L and the right subtree is of size R. The size of the tree is $1 + L + R$. For some constant C we can compute the sparsity vector of size $L+1$ for the left subtree in time $C \cdot L^2$ and the sparsity vector of size $R+1$ for the right subtree in time $C \cdot R^2$. These two sparsity vectors can be combined to get the sparsity vector for the root. As can be easily verified, combining the sparsity vectors takes time $C' \cdot LR$ for some constant C'. Thus, overall time to compute the sparsity vector for the root is

$$C \cdot L^2 + C \cdot R^2 + C' \cdot LR \leq C \cdot (1+L+R)^2$$

if $C \geq C'$ as required. □

Theorem 23. *Given a binary tree of size n and an integer k, we can solve the Tree Sparsity problem in time $O(kn)$.*

Proof. We observe that for every node we need to compute $k+1$ entries of the sparsity vector.

Consider the original tree. A node it *heavy* if both left and the right subtrees are of size $\geq k$. If both subtrees are of size $< k$, the node is *light*. Otherwise, the node is *average*.

The runtime corresponding to heavy, average and light vertices can be analyzed as follows.

- A simple counting reveals that the total number of heavy nodes is $O(n/k)$. The total runtime corresponding to the heavy nodes is $O(n/k) \cdot O(k^2) = O(nk)$, where $O(k^2)$ comes from combining two sparsity vectors of size $k+1$.

- Consider an average node u. The runtime corresponding to it is upper bounded by $O(n_u k)$ where n_u is the size of the subtree of u. This is because one sparsity vector is of size $n_u + 1$ and the other is of size $k+1$. The total runtime corresponding to the average nodes is $\sum_u O(n_u k) = O(nk)$, where the summation is over all average nodes.

- Consider a light node u whose parent is average or heavy. Let s_u be the size of the tree rooted at u. The runtime corresponding to this tree is $O(s_u^2)$ by Theorem 22. The total runtime corresponding to the light nodes is upper bounded by $\sum_u O(s_u^2) = O(nk)$ where the summation is over all light nodes whose parents are heavy or average. The inequality follows because $\sum_u s_u = O(n)$ and $s_u = O(k)$ for all such nodes u.

□