Electronic Colloquium on Computational Complexity, Report No. 124 (2024)

Solving Tree Evaluation in $o(\log n \cdot \log\log n)$ space

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science, Rehovot, Israel.

July 26, 2024

Abstract

The input to the Tree Evaluation problem is a binary tree of height h in which each internal vertex is associated with a function mapping pairs of ℓ-bit strings to ℓ-bit strings, and each leaf is assigned an ℓ-bit string. The desired output is the value of the root, where the value of each internal node is defined by applying the corresponding function to the value of its children.

A recent result of Cook and Mertz (*ECCC*, TR23-174) asserts that the Tree Evaluation problem can be solved in space $O(\ell + h \cdot \log \ell)$, where the input length is $\exp(\Theta(h+\ell))$. Building on our recent exposition of their result (*ECCC*, TR24-109), we now obtain an $o((h+\ell)\cdot\log(h+\ell))$ space bound. Specifically, for the case of $h \geq \ell$, we shave off an $\Theta(\log\log(h+\ell))$ factor.

The improvement is obtained by improving the procedure of Cook and Mertz for a generalized tree evaluation problem that refers to d-ary trees. We then reduce the binary case to the d-ary case while cutting the height of the tree by a factor of $\log_2 d$.

The main result reported in this memo was obtained by Manuel Stoeckl, a student at Dartmouth, half a year before us. Manuel felt that this result is a minor observation and did not find the time to write it down. He also declined our invitation to co-author this memo.

A suggested warm-up. This exponsition builds on our exposition of the Cook and Mertz theorem [1], which asserts that the Tree Evaluation problem can be solved in space $O(\log n \cdot \log\log n)$, where n denotes the length of the input. Hence, our exposition [3] is a good warm-up for the current memo; in particular, we suggest reading [3, Sec. 1], which focuses on proving [1, Thm. 10] (and maybe also [3, Sec. 2], which yields a proof of [1, Thm. 15]). We warn that the exposition of [3], which we shall follow, refers to a *model of global storage*, which is spelled-out in [3, Sec. 3] (following [2, Sec. 5.2.4.2]).[1]

The generalized Tree Evaluation problem ($\mathtt{TrEv}^d_{h,\ell}$). The input to this computational problem is a rooted d-ary tree of height h in which internal nodes represent arbitrary gates mapping d-tuples of ℓ-bit strings to ℓ-bit strings, and each leaf carries an ℓ-bit string. Specifically, nodes in the tree are associated with d-ary sequences of length at most h such that the nodes $u1, ..., ud$ are the d children of the node $u \in U \stackrel{\text{def}}{=} \bigcup_{i=0}^{h-1}[d]^i$. For every $u \in U$, the internal node u is associated with a gate $f_u : \{0,1\}^{d\cdot\ell} \to \{0,1\}^\ell$, and the leaf $u \in \{0,1\}^h$ is assigned the value $v_u \in \{0,1\}^\ell$.

[1] Loosey speaking, this model abandon the paradigm of "good programming" under which a recursive call uses a different work space than the execution that calls it. Instead, it uses the same *global space* for both executions, whereas only a much smaller work space will be allocated to each recursive level as its *local space*.

ISSN 1433-8092

Hence, the input is the description of all $|U| = \frac{d^h-1}{d-1}$ gates (i.e., all f_u's) and the values assigned to the d^h leaves; that is, the length of the input is $|U| \cdot (2^{d\ell} \cdot \ell) + d^h \cdot \ell = \exp(\Theta(d\ell + h \log d))$. The desired output is v_λ such that for every $u \in U$ it holds that

$$v_u = f_u(v_{u1}, ..., v_{ud}). \tag{1}$$

The Tree Evaluation problem corresponds to the special case of $d = 2$; that is, $\mathtt{TrEv}_{h,\ell} \stackrel{\text{def}}{=} \mathtt{TrEv}^2_{h,\ell}$. For the history and significance of the Tree Evaluation problem, see [1]. In particular, the generalized version ($\mathtt{TrEv}^d_{h,\ell}$) was introduced in [1, Sec. 6.1], and was shown to be solvable in space $O((h + d\ell) \cdot \log(d\ell))$ [1, Thm. 18]. Here, we slightly improve upon this bound.

Low degree extensions and interpolation. In analogy to [3, Sec. 2], we associate $\{0,1\}^\ell$ with $[dk]^k$ (equiv., $\{0,1\}^{d\cdot\ell}$ with $[dk]^{d\cdot k}$), and consider functions that desribe the individual elements in the outputs of the f_u's. Specifically, for every $u \in U$ and $i \in [k]$ (and every $x^{(1)}, ..., x^{(d)} \in [dk]^k$), let $f_{u,i}(x^{(1)}, ..., x^{(d)}) \in [dk]$ equal the i^{th} symbol of $f_u(x^{(1)}, ..., x^{(d)}) \in [dk]^k$ (i.e., $f_u(x^{(1)}, ..., x^{(d)}) = (f_{u,i}(x^{(1)}, ..., x^{(d)}), ..., f_{u,i}(x^{(1)}, ..., x^{(d)}))$). We use a finite field of size $\text{poly}(dk)$ that is greater than $m = d \cdot k^2$ (and $[dk] \subset \mathcal{K}$), and consider low degree extensions of the $f_{u,i}$'s. Specifically, for each $u \in U$ and $i \in [k]$, we let $\widehat{f}_{u,i} : \mathcal{K}^{d\cdot k} \to \mathcal{K}$ be a $d \cdot k$-variate polynomial of individual degree $k-1$ over $\mathcal{K}$ that extends $f_{u,i} : [dk]^{dk} \to [dk]$.[2] Note that $\widehat{f}_{u,i}$ has total degree $dk \cdot (k-1) < m$, whereas its input length (i.e., $\log_2 |\mathcal{K}^{dk}|$) equals $\log_2(\text{poly}(dk)^{dk}) = O(d\ell)$. The punchline is that, for every $v_1, ..., v_d \in \mathcal{K}^k$, *we can obtain the value of* $\widehat{f}_{u,i}(v_1, ..., v_d)$ *by univariate polynomial interpolation from the values of* $\widehat{f}_{u,i}(j\widehat{x}^{(1)} + v_1, ..., j\widehat{x}^{(d)} + v_d)$ *for all* $j \in [m] \subset \mathcal{K}$, where $j \cdot (z_1, ..., z_k) \in \mathcal{K}^k$ equals $(jz_1, ..., jz_k)$.

Note, however, that a naive implementation of the foregoing interpolation involves operating on these m values (after storing them in memory). Fortunately, *the interpolation formula is a linear combination of these m values, and so we need not store these values but can rather operate on them on-the-fly* (while only storing the partial linear combination computed so far). Specifically, we let c_j be the coefficient of $\widehat{f}_{u,i}(j\widehat{x}^{(1)} + v_1, ..., j\widehat{x}^{(d)} + v_d)$ used to obtain $\widehat{f}_{u,i}(0\widehat{x}^{(1)} + v_1, ..., 0\widehat{x}^{(d)} + v_d)$; that is,

$$\widehat{f}_{u,i}(v_1, ..., v_d) = \sum_{j\in[m]} c_j \cdot \widehat{f}_{u,i}(j\widehat{x}^{(1)} + v_1, ..., j\widehat{x}^{(d)} + v_d). \tag{2}$$

Then, we shall compute the r.h.s of Eq. (2) in m iterations such that in each iteration we obtain and add the current term to the partial sum computed so far.

Our (recursive) algorithm. For sake of simplicity, we first assume that we have oracle access to the function $F : U \times [k] \times \mathcal{K}^{dk} \to \mathcal{K}$ defined by

$$F(u, i, \widehat{x}^{(1)}, ..., \widehat{x}^{(d)}) \stackrel{\text{def}}{=} \widehat{f}_{u,i}(\widehat{x}^{(1)}, ..., \widehat{x}^{(d)}). \tag{3}$$

[2] Indeed, for simplicity, we assume that $\mathcal{K}$ is of prime cardinality. In general, for $S \subset \mathcal{K}$, the low degree extension of $f : S^t \to S$ is given by $\widehat{f} : \mathcal{K}^t \to \mathcal{K}$ such that

$$\widehat{f}(x_1, ..., x_t) = \sum_{a_1,...,a_t \in S} \left(\prod_{i\in[t]} \chi_{a_i}(x_i) \right) \cdot f(a_1, ..., a_t),$$

where $\chi_a(x) \stackrel{\text{def}}{=} \prod_{b\in S\setminus\{a\}} (x-b)/(a-b)$ is a degree $|S|-1$ univariate polynomial.

The global memory that we use will hold $d+1$ elements of $\mathcal{K}^k$ (each being a k-sequence over $\mathcal{K}$), denoted $\widehat{x}^{(1)},...,\widehat{x}^{(d)}$, and $\widehat{z}$, as well as a sequence (over $[d]$) of length at most h, denoted u. Now, suppose that we have a procedure that, for any $u \in U$, $\sigma \in [d]$ and $\tau \in \{0,1\}$, when invoked with $(u\sigma, \tau, \widehat{x}^{(1)},...,\widehat{x}^{(d)},\widehat{z})$ on the global space, returns $(u\sigma, \widehat{x}^{(1)},...,\widehat{x}^{(d)},\widehat{z}+(-1)^\tau \cdot v_{u\sigma})$ on the global space, where $v_{u\sigma} \in [dk]^k \subset \mathcal{K}^k$ is recursively defined as in Eq. (1).[3] Then, when invoked with $(u,\tau,\widehat{x}^{(1)},...,\widehat{x}^{(d)},\widehat{z})$ on the global space, we can return $(u,\widehat{x}^{(1)},...,\widehat{x}^{(d)},\widehat{z}+(-1)^\tau v_u)$ such that $v_u = f_u(v_{u1},...,v_{ud})$, by proceeding in m iterations as follows.[4]

> (In iteration $j \in [m]$, for each $i \in [k]$, we shall increment the current value of the i^{th} element of $\widehat{z}$ by $(-1)^\tau \cdot c_j \cdot \widehat{f}_{u,i}(j\widehat{x}^{(1)}+v_{u1},...,j\widehat{x}^{(d)}+v_{ud})$, while maintaining $(u,\widehat{x}^{(1)},...,\widehat{x}^{(d)})$ intact.)[5]
>
> The j^{th} iteration proceeds as follows.
>
> 1. Proceeding in d sub-steps (corresponding to all $\sigma \in [d]$), in the σ^{th} sub-step we place $j\widehat{x}^{(\sigma)}$ in the last position and make a recursive call aimed to increament it by $v_{u\sigma}$. That is, for $\sigma = 1,...,d$, making a recursive call with a global space containing $(u\sigma, 0, \widehat{y}^{(1)},...,\widehat{y}^{(\sigma-1)},\widehat{x}^{(\sigma+1)},...,\widehat{x}^{(d)},\widehat{z},j\widehat{x}^{(\sigma)})$, we update the global space to $(u\sigma,\widehat{y}^{(1)},...,\widehat{y}^{(\sigma-1)},\widehat{x}^{(\sigma+1)},...,\widehat{x}^{(d)},\widehat{z},\widehat{y}^{(\sigma)})$, where $\widehat{y}^{(\sigma)} \stackrel{\text{def}}{=} j\widehat{x}^{(\sigma)} + v_{u\sigma}$.
>   (Once these d sub-steps are completed, the global space contains $(u,\widehat{y}^{(1)},...,\widehat{y}^{(d)},\widehat{z})$, where $\widehat{y}^{(\sigma)} = j\widehat{x}^{(\sigma)} + v_{u\sigma}$ for every $\sigma \in [d]$.)
> 2. For each $i \in [k]$, letting $\widehat{z}_i$ denote the i^{th} element of $\widehat{z} \in \mathcal{K}^k$, compute $\widehat{z}_i + (-1)^\tau \cdot c_j \cdot F(u,i,\widehat{y}^{(1)},....,\widehat{y}^{(d)})$ by making an oracle call to F, and update the value of $\widehat{z}_i$ accordingly. Note that in the i^{th} sub-step only the i^{th} element of the sequence $\widehat{z}$ is updated (whereas multiplication by c_j is performed so to fit Eq. (2)).
> 3. Analogously to Step 1, for $\sigma = 1,...,d$, making a recursive call with a global space containing $(u\sigma,1,\widehat{x}^{(1)},...,\widehat{x}^{(\sigma-1)},\widehat{y}^{(\sigma+1)},...,\widehat{y}^{(d)},\widehat{z},\widehat{y}^{(\sigma)})$, we update the global space to $(u\sigma,\widehat{x}^{(1)},...,\widehat{x}^{(\sigma-1)},\widehat{y}^{(\sigma+1)},...,\widehat{y}^{(d)},\widehat{z},j\widehat{x}^{(\sigma)})$, since $\widehat{y}^{(\sigma)} - v_{u\sigma} = j\widehat{x}^{(\sigma)}$.
> 4. Re-arrange the global space to contain $(u,\widehat{x}^{(1)},...,\widehat{x}^{(d)},\widehat{z})$, while noting that each $\widehat{z}_i$ got incremented by $(-1)^\tau \cdot c_j \cdot \widehat{f}_{u,i}(j\widehat{x}^{(1)}+v_{u1},...,j\widehat{x}^{(d)}+v_{ud})$.
>
> Using Eq. (2), we note that (after the m iterations) the value of each $\widehat{z}_i$ equals the initial value plus $(-1)^\tau \cdot \widehat{f}_{u,i}(v_{u1},...,v_{ud})$.

Letting $\widetilde{h} \stackrel{\text{def}}{=} h \log_2 h$, the foregoing recursive procedure uses a global space of length $\widetilde{h}+O(1)+(d+1+o(1))\cdot\log_2|\mathcal{K}|^k = \widetilde{h}+O(dk\cdot\log(dk)) = h+O(d\ell)$ and a local space of length $\log_2 m = O(\log dk)$. (The $o(1)\cdot\log_2|\mathcal{K}|^\ell + O(\log d)$ term accounts for the space complexity of various manipulations (including maintaining the counters $i \in [k]$ and $\sigma \in [d]$), whereas the local space is used only for recording $j \in [m]$.)

Using a composition lemma akin [2, Lem. 5.10], it follows that the general Tree Evaluation problem (with parameters h,ℓ and d) can be solved in space $O(d\ell + h\cdot\log(d\ell))$, when using oracle

[3] The variable/parameter τ allows us to either add or subtract the value $v_{u\sigma}$. In our recursive calls, we shall need both options.

[4] The following description is for the case of $u \in U$. In case $u \in [d]^h$, we may just obtain v_u from the input oracle (e.g., augment F such that $F(u) = v_u$).

[5] Recall that, by Eq. (2), $\sum_{j\in[m]} c_j \cdot \widehat{f}_{u,i}(j\widehat{x}^{(1)}+v_{u1},...,j\widehat{x}^{(d)}+v_{ud})$ equals $\widehat{f}_{u,i}(v_{u1},...,v_{ud})$.

access to F, which in turn can be evaluated in linear space (i.e., space linear in $h + d\ell$).[6] Using a naive composition (see [3, Sec. 4] for details), it follows that

Theorem 1 (an intermediate result): *The space complexity of* $\mathtt{TrEv}^d_{h,\ell}$ *is* $O(d\ell + h \cdot \log(d\ell))$.

Theorem 1 improves over the $O((d\ell + h) \cdot \log(d\ell))$ bound given in [1, Thm. 18].

Towards improving the bound for the binary case. The bound provided by Theorem 1 suggest that it may be worthwhile to reduce $\mathtt{TrEv}_{h,\ell}$ to $\mathtt{TrEv}^{2^{h'}}_{h/h',\ell}$ by replacing binary subtrees of height h' by $2^{h'}$-ary functions. The point is that space complexity of $\mathtt{TrEv}^{2^{h'}}_{h/h',\ell}$ is $O(2^{h'} \cdot \ell + (h/h') \cdot \log(2^{h'}\ell))$, which equals $O(h + 2^{h'} \cdot \ell + (h/h') \cdot \log \ell)$. Recalling that the input to $\mathtt{TrEv}_{h,\ell}$ has length $\exp(\Theta(h + \ell))$, we infer that the complexity of $\mathtt{TrEv}_{h,\ell}$ is $O((h + \ell) \cdot (2^{h'} + \frac{\log \ell}{h'}))$, which is $O((h + \ell) \cdot \frac{\log \ell}{\log \log \ell})$ (when using $h' = 0.99 \log_2 \log \ell$).

Reducing $\mathtt{TrEv}_{h,\ell}$ to $\mathtt{TrEv}^{2^{h'}}_{h/h',\ell}$. The reduction maps the binary functions $f_u : \{0,1\}^{2\ell} \to \{0,1\}^\ell$ associated with all $u \in U \stackrel{\text{def}}{=} \bigcup_{i=0}^{h-1} \{0,1\}^i$ to functions $f'_{u'} : \{0,1\}^{2^{h'}\ell} \to \{0,1\}^\ell$ associated with all $u' \in U' \stackrel{\text{def}}{=} \bigcup_{i=0}^{(h/h')-1} [2^{h'}]^i$. Specifically, for every $u' \in [2^{h'}]^i \subset U'$ viewed as an $h' \cdot i$-bit long string (for some $i \in \{0, 1, ..., h-1\}$), we define $f'_{u'}$ as the result of a computation on the binary sub-tree of height h' that is rooted at u' (i.e., the internal node reachable by the path u'' from u' uses the function $f_{u'u''}$). The space complexity of computing each of these functions is $O(\ell + h' \cdot \log \ell)$, which is evidently dominated by $O(2^{h'} \cdot \ell)$. Hence, composing this reduction with the algorithm for $\mathtt{TrEv}^{2^{h'}}_{h/h',\ell}$, we obtain.

Theorem 2 (the main result): *For every* $h' \in [h]$, *the space complexity of* $\mathtt{TrEv}_{h,\ell}$ *is* $O(h + 2^{h'} \cdot \ell + (h/h') \cdot \log \ell)$. *In particular, the space complexity of* $\mathtt{TrEv}_{h,\ell}$ *is* $O((h + \ell) \cdot \frac{\log \ell}{\log \log \ell})$.

Recalling that the length of the input to $\mathtt{TrEv}_{h,\ell}$ is exponential in $h + \ell$ and that the upper bound provided by [1, Thm. 15] is $O(\ell + h \cdot \log \ell)$, which is optimal for $h = O(\ell / \log \ell)$. Focusing on $h = \omega(\ell / \log \ell)$, we observe that in this case Theorem 2 improves over [1, Thm. 15].

In general, letting $n = \exp(\Theta(h + \ell))$ denote the length of the input to $\mathtt{TrEv}_{h,\ell}$, we get a space bound of $O(\frac{\log n \cdot \log \log n}{\log \log \log n})$.

Acknowledgments

I am grateful to James Cook and Ian Mertz for notifying me of the fact that Manuel Stoeckl obtained the main result half a year before me.

[6] Recall that computing F calls for computing the corresponding $\widehat{f}_{u,i}$, which is a multi-linear extension of $f_{u,i}$. As for computing $\widehat{f}_{u,i}$, it requires obtaining all values of $f_{u,i}$ (cf. [3, Sec. 4]).

References

[1] James Cook and Ian Mertz. Tree Evaluation is in Space $O(\log n \cdot \log \log n)$. *ECCC*, TR23-174, 2023.

[2] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[3] Oded Goldreich. On the Cook-Mertz Tree Evaluation procedure. *ECCC*, TR24-109, 2024.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il