Oded (June 26, 2024): On the Cook-Mertz Tree Evaluation procedure

We provide an overview and a digest of the recent result of Cook and Mertz [1] that asserts that Tree Evaluation can be computed in space $O(\log n \cdot \log\log n)$. In particular, we point out that the algebraic manipulation performed in [1] is merely a special case of polynomial extrapolation, and using this observation we provide a more transparent presentation as well as a low order quantitative improvement.[1]

The Tree Evaluation problem ($\mathtt{TrEv}_{h,\ell}$). The input to this computational problem is a binary tree of height h in which internal nodes represent arbitrary gates mapping pairs of ℓ-bit strings to an ℓ-bit string, and each leaf carries an ℓ-bit strings. That is, nodes in the tree are associated with strings of length at most h such that $u0$ and $u1$ are the children of node $u \in U \stackrel{\text{def}}{=} \bigcup_{i=0}^{h-1}\{0,1\}^i$. For every $u \in U$, the internal node u is associated with a gate $f_u : \{0,1\}^{\ell+\ell} \to \{0,1\}^\ell$, and the leaf $u \in \{0,1\}^h$ is assigned the value $v_u \in \{0,1\}^\ell$. Hence, the input is the description of all $|U| = 2^h - 1$ gates (i.e., all f_u's) and the values assigned to the 2^h leaves; that is, the length of the input is $(2^h-1)\cdot(2^{2\ell}\cdot\ell)+2^h\cdot\ell = \exp(\Theta(h+\ell))$. The desired output is v_λ such that for every $u \in U$ it holds that $v_u = f_u(v_{u0}, v_{u1})$. (For the history and significance of the Tree Evaluation problem, see [1].)

The straightforward recursive algorithm. Observing that the value at node u is determined by the values at its two children, we compute v_u by first making a recursive call for the value of v_{u0} and then making a recursive call for the value of v_{u1}. Hence, before making the second recursive call, we maintain the value v_{u0} in the local memory of the current execution (which refers to node u). Once we obtain v_{u1}, we compute v_u and output it. The crucial point is that each level of recursion uses a local memory that is different from the memory that is used by other levels. Hence, the space complexity of the algorithm that unravells the recursion is $O(h \cdot \ell)$.

Towards the improved (recursive) algorithm. The first step is conceptual: It consists of abandoning the paradigm of "good programming" under which a recursive call uses a different work space than the execution that calls it. Instead, we shall use 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*. (Such a model, spelled-out in Definition 3 (below), was used by us in [2, Sec. 5.2.4.2]; the "catalytic space model" used by [1] is a special case.)

The key question is how to implement the foregoing recursion in this (global storage) model. For starters, suppose that the global memory holds three ℓ-bit strings, denoted x, y and z. Further suppose that we have a procedure that, for $u \in U$ and $\sigma \in \{0,1\}$, when invoked with $(u\sigma, x, y, z)$ on the global space returns $(u\sigma, x, y, z \oplus v_{u\sigma})$ on the global space, where $v_{u\sigma}$ is as recursively defined above. Then, when invoked with (u, x, y, z) on the global space, we can return $(u, x, y, z \oplus v_u)$ by proceeding as follows:

1. Making a recursive call on $(u0, y, z, x)$, we update the global space to $(u0, y, z, x')$, where $x' \stackrel{\text{def}}{=} x \oplus v_{u0}$.

   (Note that we re-arranged the parts of the global space so that the variable holding x is updated (to a value denoted x') and the other variables are left intact.)

[1] Referring to the parameters h and ℓ as defined below, we improve the space complexity from $O((\ell+h)\cdot\log\ell)$ to $O(\ell + h\cdot\log\ell)$.

2. Making a recursive call on $(u1, x', z, y)$, we update the global space to $(u1, x', z, y')$, where $y' \stackrel{\text{def}}{=} y \oplus v_{u1}$.

3. *Miraculously* compute $z' \stackrel{\text{def}}{=} z \oplus f_u(v_{u0}, v_{u1})$ based on $x' = x \oplus v_{u0}$ and $y' = y \oplus v_{u1}$, while preserving the values of x' and y'.

4. Making a recursive call on $(u0, y', z', x')$, we update the global space to $(u0, y', z', x)$.
   (Note that $x' \oplus v_{u0}$ equals the original value of x.)

5. Making a recursive call on $(u1, x, z', y')$, we update the global space to $(u1, x, z', y)$.

6. Return (u, x, y, z').

Indeed, the problem is with the *miraculous step* (i.e., Step 3): We wish to compute $z \oplus f_u(v_{u0}, v_{u1})$, but we don't have v_{u0} and v_{u1}, but rather versions of these values that are masked by the original values of x and y, respectively. There is hope for such a miracle only if we have a few versions of this masking. Suppose, for example, that f_u were a linear (over GF(2)) function and that we have the values of $f_u(x', y')$ and $f_u(x, y)$; then, using $f_u(x', y') \oplus f_u(x, y) = f_u(x' \oplus x, y' \oplus y)$, we can obtain $f_u(x' \oplus x, y' \oplus y) = f_u(v_{u0}, v_{u1})$, if we ignore the problem of having to store both $f_u(x', y')$ and $f_u(x, y)$. The last difficulty is not an issue if we deal with the bits of these ℓ-bit values one at a time; that is, for each $i \in [\ell]$, we first compute the i^{th} of $f_u(x', y')$ and of $f_u(x, y)$, and obtain the corresponding bit of $f_u(v_{u0}, v_{u1})$.

Multi-linear extensions and extrapolation Needless to say, we do not want to assume that the f_u's are linear. The alternative of using multi-linear extensions (of functions describing the output bits) arises naturally. Indeed, we considering multi-linear extensions of the corresponding functions, where these extensions are in a (prime) field $\mathcal{K}$ that contains at least $2\ell + 2$ elements. Specifically, for every $u \in U$ and $i \in [\ell]$, let $f_{u,i}(x, y)$ equal the i^{th} bit of $f_u(x, y)$. Next, we define $\widehat{f}_{u,i} : \mathcal{K}^\ell \times \mathcal{K}^\ell \to \mathcal{K}$ to be the multi-linear extension of $f_{u,i} : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$. Now, suppose that we are given the values of $\widehat{f}_{u,i}(j\widehat{x} + v_0, j\widehat{y} + v_1)$ for every $j \in \{1, ..., 2\ell + 1\} \subset \mathcal{K}$, where $j \cdot (z_1, ..., z_\ell) = (jz_1, ..., jz_\ell)$. Using polynomial extrapolation (on the degree 2ℓ univariate polynomial (in j) obtained by fixing $u, i, \widehat{x}, \widehat{y}, v_0$ and v_1), we obtain $\widehat{f}_{u,i}(0\widehat{x} + v_0, 0\widehat{y} + v_1)$. Note, however, that a naive implementation of this extrapolation involves operating on these $2\ell+1$ values (after storing them in memory). Fortunately, *the extrapolation formula is a linear combination of these $2\ell + 1$ 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).

Actually, as observed in [1], using specific extrapolation points allows for a more explicit extrapolation that merely sums-up the values (rather than using a more general linear combination). Specifically, these extrapolation points are powers of an m^{th} root of unity, where $m > \ell' \stackrel{\text{def}}{=} 2\ell$ and $m < |\mathcal{K}| = O(\ell)$. Denoting such a root by ω, we observe that for any multi-linear polynomial $p : \mathcal{K}^{\ell'} \to \mathcal{K}$ it holds that

$$\sum_{j \in [m]} p(\omega^j z_1 + w_1, ..., \omega^j z_{\ell'} + w_{\ell'}) = m \cdot p(w_1, ..., w_{\ell'}). \tag{1}$$

(Eq. (1) can be proved by considering each monomial separately.)[2]

The improved (recursive) algorithm. For sake of simplicity, we first assume that we have oracle access to $F : U \times [\ell] \times \mathcal{K}^{2\ell} \to \mathcal{K}$ defined by

$$F(u, i, \widehat{x}, \widehat{y}) \stackrel{\text{def}}{=} \widehat{f}_{u,i}(\widehat{x}, \widehat{y}). \tag{2}$$

The global memory that we use will hold three ℓ-long sequences over $\mathcal{K}$, denoted $\widehat{x}$, $\widehat{y}$ and $\widehat{z}$, as well as a string of length at most ℓ, denoted u. We seek to construct a recursive procedure that, for $u \in U$ and $\sigma, \tau \in \{0,1\}$, when invoked with $(u\sigma, \tau, \widehat{x}, \widehat{y}, \widehat{z})$ on the global space returns $(u\sigma, \widehat{x}, \widehat{y}, \widehat{z} + (-1)^\tau \cdot v_{u\sigma})$ on the global space, where $v_{u\sigma} \in \{0,1\}^\ell \subset \mathcal{K}^\ell$ is as recursively defined above.[3] Now, when invoked with $(u, \tau, \widehat{x}, \widehat{y}, \widehat{z})$ on the global space, we can return $(u, \widehat{x}, \widehat{y}, \widehat{z} + (-1)^\tau v_u)$ by proceeding in m iterations.[4]

> In iteration $j \in [m]$, for each $i \in [\ell]$, we increment the current value of the i^{th} element of $\widehat{z}$ by $(-1)^\tau \cdot \widehat{f}_{u,i}(\omega^j \widehat{x} + v_{u0}, \omega^j \widehat{y} + v_{u1})/m$, while maintaining $(u, \widehat{x}, \widehat{y})$ intact.
>
> Recall that, by Eq. (1), $\sum_{j \in [m]} \widehat{f}_{u,i}(\omega^j \widehat{x} + v_{u0}, \omega^j \widehat{y} + v_{u1})/m$ equals $\widehat{f}_{u,i}(v_{u0}, v_{u1})$.
>
> The j^{th} iteration proceeds as follows.
>
> 1. Making a recursive call on $(u0, 0, \widehat{y}, \widehat{z}, \omega^j \widehat{x})$, we update the global space to $(u0, \widehat{y}, \widehat{z}, \widehat{x}')$, where $\widehat{x}' \stackrel{\text{def}}{=} \omega^j \widehat{x} + v_{u0}$.
> 2. Making a recursive call on $(u1, 0, \widehat{x}', \widehat{z}, \omega^j \widehat{y})$, we update the global space to $(u1, \widehat{x}', \widehat{z}, \widehat{y}')$, where $\widehat{y}' \stackrel{\text{def}}{=} \omega^j \widehat{y} + v_{u1}$.
> 3. For each $i \in [\ell]$, letting $\widehat{z}_i$ denote the i^{th} element of $\widehat{z} \in \mathcal{K}^\ell$, compute $\widehat{z}_i + (-1)^\tau \cdot F(u, i, \widehat{x}', \widehat{y}')/m$ 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 (and that division by m compensates for the factor of m in Eq. (1)).
> 4. Making a recursive call on $(u0, 1, \widehat{y}', \widehat{z}, \widehat{x}')$, we update the global space to $(u0, \widehat{y}', \widehat{z}, \omega^j \widehat{x})$. (Note that $\widehat{x}' - v_{u0} = \omega^j \widehat{x}$.)
> 5. Making a recursive call on $(u1, 1, \omega^j \widehat{x}, \widehat{z}, \widehat{y}')$, we update the global space to $(u1, \omega^j \widehat{x}, \widehat{z}, \omega^j \widehat{y})$.

[2] For any $I \subseteq [\ell']$, it holds that

$$\begin{aligned}
\sum_{j \in [m]} \prod_{i \in I} (\omega^j z_i + w_i) &= \sum_{j \in [m]} \sum_{S \subseteq I} \left(\prod_{i \in S} \omega^j z_i \right) \cdot \left(\prod_{i \in I \setminus S} w_i \right) \\
&= \sum_{S \subseteq I} \sum_{j \in [m]} \omega^{j \cdot |S|} \left(\prod_{i \in S} z_i \right) \cdot \left(\prod_{i \in I \setminus S} w_i \right) \\
&= m \cdot \prod_{i \in I} w_i,
\end{aligned}$$

where the last equality uses $\sum_{j \in [m]} \omega^{js} = 0$ for $s \in [\ell'] \subseteq [m-1]$ and $\sum_{j \in [m]} \omega^0 = m$.

[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 $u \in \{0,1\}^\ell$, we may just obtain v_u from the input oracle (e.g., augment F such that $F(u) = v_u$).

6. Re-arrange the global space to contain $(u, \widehat{x}, \widehat{y}, \widehat{z})$, while noting that each $\widehat{z}_i$ got incremented by $(-1)^\tau \cdot \widehat{f}_{u,i}(\omega^j \widehat{x} + v_{u0}, \omega^j \widehat{y} + v_{u1})/m$.

Using Eq. (1), we note that (after the m iterations) the value of each $\widehat{z}_i$ equals the initial value plus $(-1)^\tau \cdot \widehat{f}_{u,i}(v_{u0}, v_{u1})$.

The foregoing recursive procedure uses a global space of length $\ell + O(1) + (3 + o(1)) \cdot \log_2 |\mathcal{K}|^\ell = O(\ell \log |\mathcal{K}|)$ and local space of length $\log_2 m = O(\log \ell)$. (The $o(1) \cdot \log_2 |\mathcal{K}|^\ell$ term accounts for the space complexity of various manipulations (including maintaining the counter $i \in [\ell]$), 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 Tree Evaluation problem (with parameters h and ℓ) can be solved in space $O((h + \ell) \cdot \log \ell)$, when using oracle access to F, which in turn can be evaluated in linear space (i.e., space linear in $O(\ell \log |\mathcal{K}|)$).[5] Using a naive composition, it follows that

Theorem 1 (Cook and Mertz [1]): *The space complexity of* $\texttt{TrEv}_{h,\ell}$ *is* $O((h + \ell) \cdot \log \ell)$.

Recalling that the length of the input to $\texttt{TrEv}_{h,\ell}$ is exponential in $h + \ell$, it follows that $\texttt{TrEv}_{h,\ell}$ is solved in space $O(\log n \cdot \log \log n)$, where $n = \exp(\Theta(h + \ell))$.

Digest and beyond

As hinted above, we believe that the model of global storage (as outlined in [2, Def. 5.8] and reproduced below) is more flexible and intuitive than the model of catalytic strorage used in [1], which may be viewed as a special case.

As hinted above, the extrapolation formula given in Eq. (1), which relies on an m^{th} root of unity, is inessential for the proof of Theorem 1. More generally, recalling that the $\widehat{f}_{u,i}$'s are polynomials of total degree 2ℓ, we can use polynomial extrapolation based on any $2\ell' + 1$ points, while noting that any such extrapolation can be represented by a linear combination of the function values (with coefficients that depend on the given points and the desired point). The only advantage of using Eq. (1) is that the extrapolation formula is a simple sum (i.e., all coefficients are 1).

Capitalizing on the last paragraph, we can reduce the length of global storage used by the recursive procedure from $O(\ell \log \ell)$ to $O(\ell)$. This can be done by viewing the $f_{u,i}$'s as (Boolean) functions over $[k]^k \times [k]^k$, where $k^k = 2^\ell$ (i.e., $k = \Theta(\ell / \log \ell)$), and using low degree extensions of these $f_{u,i}$'s. Specifically, these extensions are $2k$-variate polynomials of individual degree $k - 1$ over $\mathcal{K} \supset [k]$, where $\mathcal{K}$ is a finite field of size $\text{poly}(k)$ that is greater than $m = 2k^2$ (and $[k] \subset \mathcal{K}$).[6] That is, $\widehat{f}_{u,i} : \mathcal{K}^k \times \mathcal{K}^k \to \mathcal{K}$ has total degree $2k \cdot (k - 1) < m$, whereas its input length (i.e., $\log_2 |\mathcal{K}^{2k}|$)

[5] Recall that computing F calls for computing the corresponding $\widehat{f}_{u,i}$, which is a multi-linear extension of $f_{u,i}$. Hence, computing $\widehat{f}_{u,i}$ requires obtaining all values of $f_{u,i}$ (compare Footnote 6).

[6] Indeed, for simplicity, we assume that $\mathcal{K}$ is of prime cardinality. In general, for $S \subset \mathcal{K}$, the low degree extension of $f : S^t \to \{0,1\}$ 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.

equals $\log_2(\mathrm{poly}(k)^{2k}) = O(\ell)$. Consequently, the revised recursive procedure uses a global space of length $O(\ell)$ and local space of length $\log_2 m = O(\log \ell)$. Hence, we obtain

Theorem 2 (an improvement over [1]): *The space complexity of* $\mathtt{TrEv}_{h,\ell}$ *is* $O(\ell + h \cdot \log \ell)$.

Hence, for $h = O(\ell/\log \ell)$, the problem can be solved in logarithmic space (because, in this case, the input length is $n = \exp(\Theta(\ell))$, whereas $O(\ell + h \cdot \log \ell) = O(\ell) = O(\log n)$).

The global storage model (mainly reproduced from [2, Sec. 5.2.4.2])

(This model was introduced in [2, Sec. 5.2.4] in order to facilitate a modular presentation of Reingold's UCONN algorithm [3].)

The aim of this model is to support a composition result that is beneficial in the context of recursive calls. The basic idea is deviating from the paradigm that allocates *separate* input/output and query devices to *each level in the recursion*, and combining all these devices in a single ("global") device, which will be used by all levels of the recursion. That is, rather than following the "structured programming" methodology of using locally designated space for passing information to the subroutine, we use the "bad programming" methodology of passing information through global variables. (As usual, this notion is formulated by referring to the model of multi-tape Turing machine, but it can be formulated in any other reasonable model of computation.)

Definition 3 (following [2, Def. 5.8]): *A* global-tape oracle machine *is defined as an oracle machine* (cf. [2, Def. 1.11])*, except that the input, output and oracle tapes are replaced by a single* global-tape*. In addition, the machine has a constant number of work tapes, called the* local-tapes*. The machine obtains its input from the global-tape, writes each query on this very tape, obtains the corresponding answer from this tape, and writes its final output on this tape.* (We stress that, as a result of invoking the oracle f, the contents of the global-tape changes from q to $f(q)$.)[7] *In addition, the machine can use the global-tape also for its internal computations. The space complexity of such a machine is stated when referring separately to its use of the global-tape and to its use of the local-tapes.*

Note that in our presentation of Theorem 1 we used oracle calls to a function F. This was done for the sake of simplicity, and these oracle calls (unlike the recursive calls) can be modeled by the usual mechanism (of oracle tapes).

References

[1] James Cook and Ian Mertz. Tree Evaluation is in Space $O(\log n \cdot \log \log n)$. *ECCC*, TR23-174, 2013.

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

[7] This means that the prior contents of the global-tape (i.e., the query q) is lost (i.e., it is replaced by the answer $f(q)$). Thus, if we wish to keep such prior contents then we need to copy it to a local-tape. We also stress that, according to the standard oracle invocation conventions, the head location after the oracle responds is at the left-most cell of the global-tape.

[3] Omer Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.