Branching Programs: Avoiding Barriers

Stephen Cook

Barriers Workshop
Princeton
August, 2009

Joint work with

Mark Braverman
Pierre McKenzie
Rahul Santhanam
Dustin Wehr

Complexity Classes

$$\mathbf{AC}^0(6) \subseteq \mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{LogCFL}$$
$$\subseteq \mathbf{AC}^1 \subseteq \mathbf{NC}^2 \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PH}$$

As far as is known, $\mathbf{AC}^0(6)$ cannot determine whether a majority of its input bits are ones.

Yet it is open whether $\mathbf{AC}^0(6) = \mathbf{PH}$.

Here we introduce the

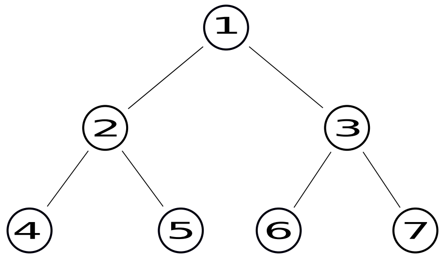
Tree Evaluation Problem (TEP)

We show TEP is in $\mathbf{LogDCFL}$.

We are trying to prove TEP $\notin \mathbf{L}$
(and TEP $\notin \mathbf{NL}$)

Tree Evaluation Problem

(Generalizes a problem in [Taitslin05])



Tree of height $h = 3$ with heap numbering

T_d^h: Balanced d-ary tree of height h
DEFAULT: $d = 2$
$[k] = \{1, ..., k\}$

TEP(h, k) Applies to T_2^h. Assume $h, k \geq 2$
Input:
$v_i \in [k]$ for each leaf i
Function $f_i : [k] \times [k] \to [k]$ for each internal node i
(Thus every node i gets a value $v_i \in [k]$)

Output: root value $v_1 \in [k]$

Decision Problem: Does $v_1 = 1$?

Claim: TEP(h, k) $\in \mathbf{LogDCFL}$

Space-efficient algorithms for TEP come from pebbling (and depth-first search)

Deterministic algorithms come from 'black' pebbling. [Paterson/Hewitt70]

Rules:

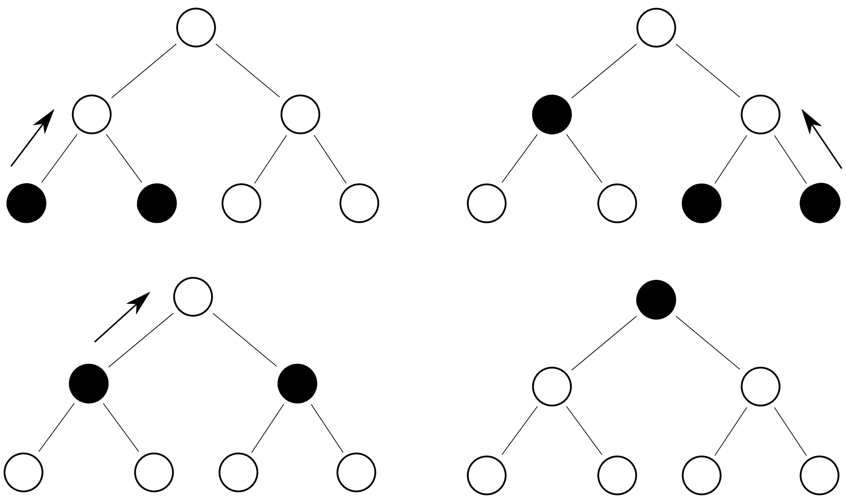
• Place a pebble on any leaf.

• If both children of node i are pebbled, slide one of them to the parent.

• Remove any pebble at any time.

Goal: Pebble the root using a minimum number of pebbles.

Easy Theorem: T_2^h requires exactly h pebbles.



Easy Theorem: T_2^h requires exactly h pebbles.

Proof:
Upper Bound: Induction on h.

Easy Theorem: T_2^h requires exactly h pebbles.

Proof:
Upper Bound: Induction on h.

Lower Bound: Every pebbling of T_2^h has a

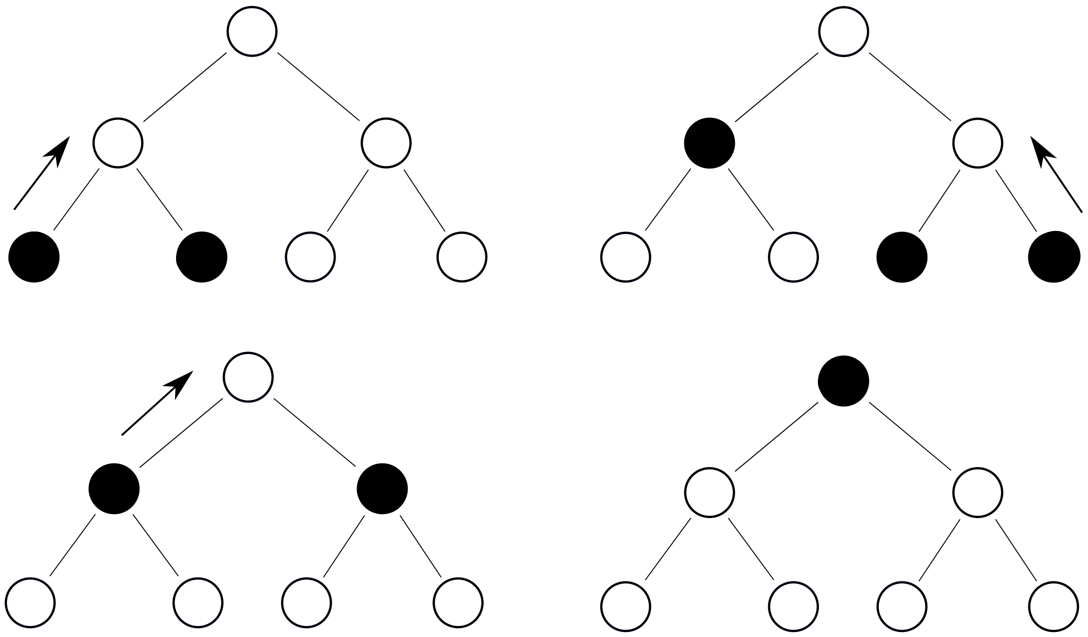
Bottleneck Configuration: The pebble set right after the first pebble move that blocks all paths from root to leaf.

This pebble move must be on a leaf ℓ

The path p from root to ℓ is pebble-free.

For each node $i \neq \ell$ on p there must be a distinct pebble blocking some path from i to a leaf.

These $h-1$ pebbles together with the pebble on ℓ form the Bottleneck.



Recall: T_2^h requires exactly h pebbles.

Corollary: $\mathsf{TEP}(h,k) \in \mathsf{DSPACE}(h \log k)$

This is NOT a log space algorithm.

Input size $n = (2^h - 1)k^2 \log k$
$\log n = \Theta(h + \log k)$

k-way Branching Programs

A k-way BP B solving TEP(h,k) is a directed multigraph with nodes called *states*. Each non-final state q is labeled either with a leaf node i, with k outedges from q labeled $1,...,k$ indicating the possible values for v_i, or labeled with (i,a,b) where i is an internal node and $a,b \in [k]$ and the outedges are labeled with the possible values for $f_i(a,b)$. Each final state has a label from $[k]$ indicating the output v_1.

$Size(B)$ is the number of states in B.

A Turing machine M solving TEP(h,k) in space $s(h,k)$ can be simulated by a family of BPs of size $2^{O(s(h,k))}$ (the number of possible configurations of M).

$Size(h, k)$ is the number of states in the smallest deterministic BP solving TEP(h, k).

$Size_h(k) = Size(h, k)$ for fixed h.

Lemma $Size_h(k) = O(k^h)$
Proof: h pebbles suffice to pebble T_2^h, and for fixed h, the number of steps in the pebbling of T_2^h is constant.

This is the best upper bound known for the order of $Size_h(k)$.

(We offer a prize to anyone who can beat it: Later)

Lemma: A lower bound of $Size_h(k) = \Omega(k^{r(h)})$ for some unbounded function $r(h)$ implies $L \neq \mathbf{LogDCFL}$.

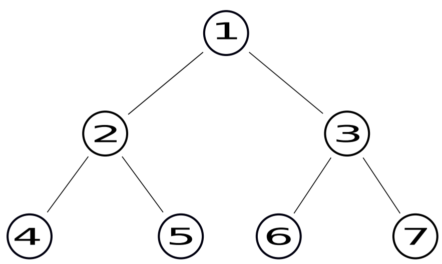
Recall best known upper bound:
$Size_h(k) = O(k^h)$

Best known lower bounds:
$Size_h(k) = \Omega(k^3)$ for each $h \geq 3$.
(Tight bounds are known for $h = 2$ and $h = 3$)

$h = 2$: $Size_2(k) = \Omega(k^2)$
This is obvious because each state of the BP can only make one query of the form (i, x, y), and there are k^2 possible values for (x, y).

$h = 3$: $Size_3(k) = \Omega(k^3)$
This is *not* obvious, because the number of input variables is only $O(k^2)$.
Proof I: Use Nečiporuk's method
Proof II: Use the "state sequence" method.

Nečiporuk's method counts the number of BPs on s states and compares this with the number of functions obtainable by various restrictions of $\mathrm{TEP}_h(k)$. This method cannot beat $\Omega(n^2)$ states, and so cannot show $\mathrm{TEP} \notin \mathbf{L}$.



Theorem: $Size_3(k) \geq k^3$

Proof: ("State Sequence" method)
For $r, s \in [k]$ let $E^{r,s}$ be the set of inputs I s.t.
- $f_1^I(x, y) = (x + y) \bmod k$
- $f_2^I(x, y) = f_3^I(x, y) = 0$ for all $(x, y) \neq (r, s)$
- $v_4^I = v_6^I = r$ and $v_5^I = v_7^I = s$

Thus $|E^{r,s}| = k^2$ because each $I \in E^{r,s}$ determined by v_2^I, v_3^I.

Let $\Gamma^{r,s}$ be the set of states which query either $f_2(r, s)$ or $f_3(r, s)$. It suffices to show
(*) $|\Gamma^{r,s}| \geq k$ for all $r, s \in [k]$.

Proof of (*): (γ^I, v_i^I) determines the output of $\mathcal{C}(I)$ (the computation on input I), where γ^I is the last state of $\mathcal{C}(I)$ in $\Gamma^{r,s}$, and i is the node queried by γ^I.

Thrifty Branching Programs

A deterministic BP solving $\mathrm{TEP}_h(k)$ is *thrifty* if for every query $f_i(x, y)$ (for every input), (x, y) are the values of the children of node i.

Thrifty BPs can implement black pebbling, and hence solve $\mathrm{TEP}_h(k)$ with $O(k^h)$ states. It turns out that this is also a lower bound.

Theorem: Thrifty deterministic BPs solving $\mathrm{TEP}_h(k)$ have $\Omega(k^h)$ states.

The proof is nontrivial. (Later)

Thus any BP beating the $O(k^h)$ upper bound must make queries $f_i(x, y)$ which are irrelevant to the value v_i of the node i.

Thrifty Hypothesis: Thrifty BPs are optimal among deterministic BPs solving $\mathrm{TEP}_h(k)$.

(Not quite true for solving the decision version of TEP)

Theorem: Thrifty deterministic BPs solving $\mathsf{TEP}_h(k)$ have $\Omega(k^h)$ states.

Proof Idea: Suppose B is a deterministic BP which solves $\mathsf{TEP}_h(k)$. We will associate with the computation $Comp_B(I)$ for each input I a black pebbling sequence for the tree $T = T_2^h$.

Trouble: The pebbling can be different for each input I.

We will show how to find a "bottleneck configuration" in the computation, similar to the one that comes out of any black pebbling of the tree T_2^h.

Let v_i^I be the value of node i for input I. We say state q *queries* v_i^I if it queries $f_i(v_{2i}^I, v_{2i+1}^I)$. Note that every node of T must be queried at least once during the computation, and for each time that a node is queried there must be earlier times at which each of its children is queried.

For each input I and each node i in T we associate a *critical* state q_i^I which queries v_i^I during the computation of B on input I as follows. Let q_1^I be the first state which queries the root v_1^I. In general, for each node i let q_{2i}^I (resp. q_{2i+1}^I) be the last state before q_i^I that queries v_{2i}^I (resp. v_{2i+1}^I).

Note that for each path from root to leaf in T the critical states are reached at decreasing times.

Pebbling Sequence Associated with input I: For each critical state q_i^I, slide one of the pebbles on the children of i (both children must be pebbled) to node i.

This is a valid black pebbling of T, because the critical states for the children of each node i precede the critical state of i.

The *bottleneck state* q_b^I of the computation is the first critical state after which all paths from root to leaf are blocked by pebbles in the associated pebbling sequence. The pebbled node b is the *bottleneck leaf*. The *bottleneck path* is the path from the root to b. The *bottleneck configuration* C^I is the set of pebbled nodes right after q_b^i. Thus

$$|C^I| \geq h$$

Define the *supercritical state* q_c^I of the computation to be the critical state at which the parent c of the bottleneck leaf b is queried. Thus c is on the bottleneck path, and all bottleneck nodes are pebbled before q_c^I.

We want a lower bound on the number of distinct supercritical states of B as I ranges over all inputs. Intuitively a supercritical state "knows" the values of all the pebbled nodes in the bottleneck configuration C^I, since (by definition of critical state) the computation queries each parent of a bottleneck node before it queries the node again.

But we must account for the possibility that the computation might gain information about these values by making queries to other nodes. We do this by partitioning the inputs into blocks such that the inputs in each block have the same node values for all nodes other than those in the bottleneck configuration C^I.

Now we restrict attention to the set E of inputs I such that the root function f_1 is identically 0, and the functions associated with all other internal nodes are 0 except possibly when evaluated at the values of their children. Thus each input I in E is determined by the values $v_2^I, v_3^I, \ldots v_n^I$ of all nodes in T other than the root, so

$$|E| = k^{n-1} \tag{1}$$

where $n = 2^h - 1$ is the number of nodes in T.

Let E_1 be the set of all inputs $I \in E$ such that $|C^I| = h$. Let $E_2 = E \setminus E_1$.

The theorem follows from

Claim: The number of distinct supercritical states is at least

(1) $(|E_1|/|E|)k^h$ for $I \in E_1$

(2) $\epsilon_h(|E_2|/|E|)k^{h+1}$ for $I \in E_2$

Proof of (1): The number of distinct supercritical states is at least $(|E_1|/|E|)k^h$, for inputs in E_1. (Here $|C^I| = h$.)

For $I \in E_1$ all h nodes in the bottleneck configuration C^I are completely determined by the supercritical node $c(I)$. Namely C^I consists of the siblings of each node (except the root) on the bottleneck path, together with the leaf b of the path.

For each node j at level 2 (just above the leaves) in T, let $C(j)$ be the bottleneck set of size h associated with the node j as above, assuming j is supercritical. Let E_1^j be the set of all inputs I in E_1 whose supercritical node $c(I) = j$. Thus the sets $\{E_1^j\}$ form a partition of E_1, and so

$$|E_1| = \sum_j |E_1^j| \qquad (2)$$

Let U be the set of nodes in T other than the root. By a counting argument, for each node j at level 2 there must be some assignment of values V_j to the the $n-h-1$ nodes in $U-C(j)$ such that there are at least $|E_1^j|/k^{n-h-1}$ inputs I in E_1^j whose nodes have the values V_j. Let $E_1^{j,*}$ be that set of inputs I. Thus

$$|E_1^{j,*}| \geq |E_1^j|/k^{n-h-1} \tag{3}$$

and each $I \in E_1^{j,*}$ is uniquely determined by the values of the h nodes in $C(j)$.

Let Q_j be the set of all states q in B such that $q = q_j^I$ for some $I \in E_1^{j,*}$. Note that the sets Q_j and $Q_{j'}$ are disjoint for distinct j, j' since the state q_j^I queries node j.

FACT: $|Q_j| = |E_1^{j,*}|$. The Lemma follows, since if Q is the set of all states in B, then by (1),(2), and (3) we have

$$\begin{aligned}|Q| \geq \sum_j |Q_j| = \sum_j |E_1^{j,*}| \geq \sum_j |E_1^j|/k^{n-h-1} \\ = |E_1|/k^{n-h-1} = (|E_1|/|E|)k^h\end{aligned}$$

The second part of the **Claim:**

Proof of (2): The number of distinct super-critical states is at least $\epsilon_h(|E_2|/|E|)k^{h+1}$, for inputs I in E_2. (Here $|C^I| \geq h+1$.)

The proof is similar to (1), but this time we partition E_2^j according to the set S consisting of the next $h-1$ bottleneck nodes which are determined after the children of the supercritical node are determined.

ϵ_h is the reciprocal of then number of such sets S.

Nondeterministic Branching Programs

Black/White Pebbling: A white pebble can be placed on any node at any time (representing a guess as to the value). The pebble can be removed if the node is a leaf, or both children have pebbles.

T_2^h can be B/W pebbled with $\lceil h/2 \rceil + 1$ pebbles. (This is optimal.)

Recall T_2^h requires h pebbles to black pebble it.

Nondeterministic Branching Programs

Black/White Pebbling: A white pebble can be placed on any node at any time (representing a guess as to the value). The pebble can be removed if the node is a leaf, or both children have pebbles.

T_2^h can be B/W pebbled with $\lceil h/2 \rceil + 1$ pebbles. (This is optimal.)

Recall T_2^h requires h pebbles to black pebble it.

Nondeterministic BPs implement B/W pebbling, so $NSize_h(k) = O(k^{\lceil h/2 \rceil + 1})$.

For $h = 3$ this gives $O(k^3)$ states, but best lower bound is $k^{2.5}$ states (via both Nečiporuk and 'state-sequence' methods).

This led us to discover "fractional pebbling".

T_2^3 can be B/W pebbled with 2.5 pebbles, so

$NSize_3(k) = \Theta(k^{2.5})$.

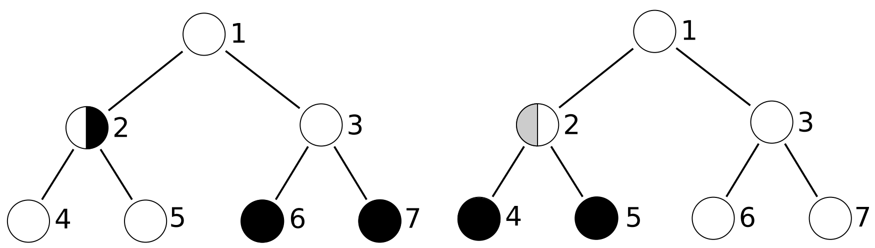
Fractional Pebbling

Fractional pebbling is like B/W pebbling, except now a node i can have a pair $(b(i), w(i))$ of real values, where

$$0 \leq b(i), w(i) \qquad b(i) + w(i) \leq 1$$

If both children of node i have total pebble value 1, then $w(i)$ can be set to 0, and any black fraction can be slid up from the children to increase $b(i)$.

The tree T_2^3 can be fractionally pebbled with 2.5 pebbles.



Theorem Thrifty nondeterministic BPs can implement fractional pebbling to solve $\mathsf{TEP}_h(k)$.

Theorem Bounds on fractional pebbling.
#FRpebbles(T_2^3) $= 2.5$
#FRpebbles(T_2^4) $= 3$
$h/2 - 1 \leq$ #FRpebbles(T_2^h) $\leq h/2 + 1$

Theorem(Repeat) Thrifty nondeterministic BPs can implement fractional pebbling.

Corollary $NSize_3(k) = \Theta(k^{2.5})$
$NSize_4(k) = O(k^3)$
$NSize_h(k) = O(k^{h/2+1}), h \geq 2$
(All upper bounds use thrifty BPs)

Theorem $ThriftyNSize_4(k) = \Theta(k^3)$

Open Question: Can nondeterministic Thrifty BPs beat fractional pebbling bound for $h > 4$?

(Recall that black pebbling is optimal for deterministic thrifty BPs.)

Previous Work

for BP size lower bounds for ‘explicit’ functions

Bor Razb Smo 93 Exp size Lower Bound for Syntactic Read k BPs

Beame Saks Sun Vee 02 (Improving on Ajtai 99): Exp size lower bound when time is restricted to $o(n\sqrt{\log n/\log\log n})$

Gal Koucky McK 08: Exp size lower bound for Incremental BPs

Neciporuk Barrier: $n^2/\log n$ size lower bound for unrestricted BPs.

Conclusion

Thrifty Hypothesis: Thrifty BPs are optimal among deterministic k-way BPs solving $\mathsf{TEP}_h(k)$.
(i.e. $Size_h(k) = \Omega(k^h)$.)

In other words, the black pebbling method is the most space-efficient deterministic method for solving $\mathsf{TEP}_h(k)$.

A proof implies $\mathbf{L} \neq \mathbf{LogDCFL}$
(so $\mathbf{NC}^1 \subsetneq \mathbf{NC}^2$).

A disproof would involve a new space-efficient algorithm and would also be interesting
(think superconcentrators).

Next Step: Prove or disprove $Size_4(k) = \Omega(k^4)$

(Best known bound: $Size_4(k) = \Omega(k^3)$.)

Are there any Barriers??

\$100 PRIZE!!!

We, the five authors listed below, offer a prize of \$100 US to the first person who, for some height $h \geq 4$ and $\epsilon > 0$, proves the existence of a family $\{B_k; k \geq 2\}$ of deterministic k-way branching programs such that B_k solves $FT_2^h(k)$ and has $O(k^{h-\epsilon})$ states.

Mark Braverman, Stephen Cook, Pierre McKenzie, Rahul Santhanam, Dustin Wehr

NOTE: Pebbling gives an upper bound of $O(k^h)$ states.

FULL PAPER (45 pages) Available on my web site.