High Parallel Complexity Graphs and Memory-Hard Functions

Joël Alwen*, Vladimir Serbinenko*

IST Austria[1], Google Zürich[2]
jalwen@ist.ac.at, phcoder@gmail.com

Abstract. We develop new theoretical tools for proving lower-bounds on the (amortized) complexity of functions in a parallel setting. We demonstrate their use by constructing the first provably secure Memory-hard functions (MHF); a class of functions recently gaining acceptance in practice as an effective means to counter brute-force attacks on security relevant functions.

Pebbling games over graphs have proven to be a powerful abstraction for a wide variety of computational models. A dominant application of such games is proving complexity lower-bounds using "pebbling reductions". These bound the complexity of a given function f_G in some computational model $\mathcal{M}$ of interest in terms of the pebbling complexity of a related graph G, where the pebbling complexity is defined via a pebbling game $\mathcal{G}$. In particular, finding a function with high complexity in $\mathcal{M}$ is reduced to (the hopefully simpler task of) finding graphs with high pebbling complexity in $\mathcal{G}$. We introduce a very simple and natural pebbling game $\mathcal{G}_p$ for abstracting parallel computation models. As an important conceptual contribution we also define a natural measure of pebbling complexity called *cumulative complexity* (CC) and show how it overcomes a crucial shortcoming in the parallel setting exhibited by more traditional complexity measures used in past reductions. Next (analogous to the results of Tarjan et. al. [PTC76,LT82] for sequential pebbling games) we demonstrates, via a novel and non-tibial proof, an explicit construction of a constant in-degree family of graphs whose CC in $\mathcal{G}_p$ approaches maximality to within a polylogarithmic factor for any graph of equal size.

To demonstrate the use of these theoretical tools we give an example in the field of cryptography, by constructing the first provably secure Memory-hard function (MHF). Introduced by Percival [Per09], an MHF can be computed efficiently on a sequential machine but further requires the same amount of memory (or much greater time) amortized per evaluation on a parallel machine. Thus, while a say an ASIC or FPGA may be faster than a CPU, the increased cost of working memory negates this advantage. In particular MHFs crucially underly the ASIC/FPGA-resistant proofs-of-work of the crypto-currency Litecoin[Cha11], the brute-force resistant key-derivation function `scrypt` [Per09] and can be used to increase the cost to an adversary of recovering passwords from a compromised login server. To place these applications on more sound theoretic footing we first provide a new formal definition (in the Random Oracle Model) and an accompanying notion of amortized memory hardness to capture the intuitive property of an MHF (thereby remedying an important issue with the original definition of [Per09]). Next we define a mapping from a graph G to a related function f_G. Finally we prove a pebbling reduction bounding the amortized memory hardness per evaluation of f_G in terms of the CC of G in the game $\mathcal{G}_p$. Together with the construction above, this gives rise to the first provably secure example of an MHF.

* The work was done primary while both authors were at ETH Zürich. This work of the first author was also partly funded by the European Research Council under an ERC Starting Grant (259668-PSPC).

1 Introduction

Since its inception by Hewitt and Paterson [HP70] and Cook [Coo73] the standard (black) pebbling game and its derivatives have proven to be extremely useful abstractions in computer science. For example the black pebbling game underlies the proof of Hopcraft, Paul and Valiant [HPV77] showing that a multitape TM with deterministic time $t(n)$ can be simulated on a (standard) TM in deterministic space $t(n)/log(t(n))$. In the context of code optimization, the game was used by Sethi [Set75] to show that determining if a given program can be executed using k registers is NP-complete.

On the highest level, the standard pebbling game over a directed acyclic graph (DAG) can be thought of as a game in which pebbles are placed on and removed from the nodes of a given DAG G in a sequence of steps according to the following simple rules. A node may contain at most one pebble at a time and the ultimate goal is to having placed a pebble, at least once, on each node from a set of target nodes (for example all sink nodes of G[1]).

1. A pebble can be placed on a node only if all of it's parents already contain a pebble at the end of the previous step. In particular a pebble may always be placed on nodes with in-degree 0.
2. A pebble can be removed from G at any time.

A given execution of the game (called a *pebbling of* G) is assigned a cost and so the complexity of G is the minimal cost of any legal and complete pebbling of G. A common cost measure is the S-cost where S is the maximum number of simultaneous pebbles on G during any step of the pebbling. Alternatively the ST-cost is the product of the maximum number of simultaneously used pebbles and the number of steps needed to complete the pebbling. More generally, one can consider the trade-off between the values S and T say by describing the minimal value of T as a function of S.

With this in mind, the pebbling paradigm can now be described in terms of three steps.

1. Initially a "real world" computational model $\mathcal{M}$ is fixed[2] and the associated cost of a given computation in $\mathcal{M}$ is defined.[3]
2. Next an idealized computational model is formalized by specifying the rules governing a pebbling game over DAGs and an associated cost is fixed. In particular this gives rise to a precise notion of the complexity of a DAG.
3. Finally, for a function f_G, related to some DAG G,[4] a reduction is given showing that f_G can computed at no less cost in $\mathcal{M}$ then (some function of) the complexity of G.

The power of this approach is that, analysing the computational complexity of such functions in $\mathcal{M}$ is reduced to the (presumably significantly easier) task of analysing the pebbling complexity of DAGs.

Memory-Hardness. A *memory-hard function (ensemble)* (MHF) is a ensemble $F = \{f_n : n \in \mathbb{N}\}$ of families of hard to compute functions $f_n = \{g_{n,i}\}$ equipped with a hardness parameter n. The precise notion of computational hardness is motivated by exploiting the real-world asymmetry between the large (monetary) cost of working memory for circuits and the (comparatively) low cost for general purpose computers. More formally this asymmetry can be viewed as the difference between measuring turing machine efficiency in terms of runtime (assuming polynomially bounded memory usage) while VLSI efficiency is commonly

[1] A sink node is a node with no outgoing edges.

[2] Say oracle-machines, TM, register-machines, pairs of ITMs or interactive RAMs for example.

[3] Say the number oracle calls, amount of work tape used, computational steps performed by the ITM, the communication complexity between the pair of ITMs, or the amount of storage accessed by the RAM.

[4] Commonly G describes the dependencies between inputs, intermediate values and the outputs involved in computing f_G.

measured in terms of AT-complexity [Tho79]; i.e. the product of runtime *and area* of the circuit. The concept was first introduced by Percival [Per09] with the application of password hashing in mind. In order to mitigate the cost of an attacker breaking into a login server and stealing the credential files, Percival suggests instead of storing a user's password p, an MHF can be used by a login server to store (n, i, z) where i is a fresh random index and $z := g_{n,i}(p)$. That way, when an adversary learns (n, i, z), the cost of brute-forcing $g_{n,i}$ to recover p from z could be scaled according to n as chosen by the login server.[5]

Starting from the observation that, in practice, while login servers are usually implemented using general purpose computers, the most effective computational environments used to mount brute-force attacks (FPGAs, ASICs [Fou98] and GPUs [Gos12]) are highly parallel circuits, Percival proposes the following notion of hardness for an MHF. A ensemble F is an MHF if for any $n \in \mathbb{N}$ and any parallel algorithm[6] computing $g_{n,i}$ (on any input and random i) using memory M and work (time) W it holds that $M * W \in \Omega(n^2)$ and moreover $g_{n,i}$ can always be computed with work $W \in O(n)$ even by a sequential algorithm. The (somewhat implicit) reasoning for this is that repetitively evaluating functions in the ensemble, even with a massively parallel circuit (but in a bounded amount time) still requires a large amount of (expensive) memory making circuit based brute-force attacks less economical. More formally while the runtime of a TM evaluating $g_{n,i}$ is $O(n)$ the AT-complexity of a circuit-based brute-force attack scales in $O(n^2)$ per password attempt.

The candidate function proposed in [Per09] has since seen a some success in practice. For example it has been used as a core building block to construct Proofs-of-Effort (PoE); that is an interactive proof systems where a prover can only convince a verifier to accept if the prover has exerted a certain amount of computational effort during the protocol execution. Indeed such a construction (where the effort takes the form of repeatedly evaluating the underlying MHF) underlies both of the crypto-currencies Litecoin [Cha11] and Dogecoin [Pal13] and the more general P2P secure public ledger [BDLHA13]. [7].

Unfortunately it turns out that formalization of memory hardness used in [Per09] is inadequate for both password storage and PoE. In particular the notion does not, in general, scale well with repeated evaluations of $g_{n,i}$ on distinct inputs (which is precisely the task of the adversary in a brute-force attack and the PoE constructions). Intuitively the shortcoming can be understood via the following example. Suppose computing $g_{n,i}$ initially (but briefly) requires $O(n)$ memory followed by a phase requiring $O(n)$ sequential work but very little memory. The result is that f_n is memory-hard according to the formalization of [Per09]. However in a parallel environment once the memory intensive phase of a first evaluation has been completed a second evaluation of $g_{n,i}$ (on new input) can begin using the now available space long before the first computation is actually complete. Using such a pipelining approach we see that the *amortized* hardness $g_{n,i}$ may actually be much less then the hardness of a single copy of $g_{n,i}$. Never the less the motivation (of forcing large *amortized* AT-complexity relative to sequential runtime) remains of interest.

1.1 Our Contribution

An important property of (most variants of) the standard pebbling game is that they require that at most one pebble be touched per step. Intuitively this restricts the applications of such games to sequential computational models. However motivated the ever increasing prevalence of parallelism in modern computational systems (e.g. GPUs, custom circuits, multi-core CPUs, cloud computers, etc.) the goals of this work are two fold. First, we provide simple and intuitive but powerful tools for applying the pebbling paradigm to parallel

[5] Here it is assumed that the most efficient way of inverting $g_{n,i}$ for relatively low entropy passwords is to simply brute-force the function until a preimage is found; an assumption which seems to hold in practice.

[6] That is an algorithm able to perform multiple computational steps in parallel.

[7] In fact, essentially the same transformation but for a different type of moderately hard function also underlies the PoE in the crypto-currency Bitcoin [Nak09] as well as the provably secure PoE constructions of [DGN03,DNW05])

settings, especially when concerned with repeated or composed computation. Second, we demonstrate these tools by realizing a new ensemble of provably secure (and formally robust) MHF functions.

We begin by modifying the rules of the standard pebbling game to obtain a very natural parallelised generalization. In particular we allow the rules to be applied batch-wise removing the restriction on the number of pebbles touched per step. Next we we introduce a new more fine-grained cost measure called the *cumulative complexity* (CC) of a graph. Put simply the CC of a given execution in the (parallel) pebbling game for graph G is the sum of the number of pebbles lying on G when summed across all steps in the execution.

To motivate the new definition in the parallel setting we show that, in contrast to both S-complexity and ST-complexity, the CC of a graph consisting of several disconnected components is equal to the sum of the CC of each of it's components. In particular, as we show later, this makes CC a much more useful tool for analysing the cost of evaluating several hard functions in parallel. This takes on special importance when considering the amortized cost of repeated function evaluation in parallel models. In contrast for any $m \geq 1$ we give a DAG G_m of size $\Theta(m^2)$ for which the parallel ST-complexity of $\Theta(m)$ copies of G is essentially the same as the parallel ST-complexity of a single copy of G

Next we provide some upper and lower bounds for the CC of specific classes of DAGs. A trivial algorithm shows that no DAG of size n can have parallel CC (nor ST-complexity) greater then n^2. Moreover if no restrictions are place on the in-degree of nodes then a trivial construction essentially matches this bound. However many interesting past applications of the pebbling paradigm required graphs to have low (usually constant) in-degree.[8] Therefore, henceforth we restrict ourselves to constant in-degree graphs.

To further motivate our new construction we look at some known constructions with extreme time/memory requirements in the sequential setting. We start with a family of DAGs consisting of stacks of superconcentrators which, in the sequential setting, exhibit an extreme trade-off between S and T [LT82]. That is even if just a few less than n pebbles are used the required (sequential) time grows exponentially. However a simple observation about the limits of the CC of a depth d graph combined with the linear-superconcentrator construction of [Pip77] show that in the parallel setting these stacks can have CC as low as $O(n \log^2(n))$. In a similar spirit we look at bit-reversal graphs which are known to enjoy optimally high sequential ST-complexity of $O(n^2)$ [LT82]. We demonstrate a parallel pebbling algorithm with a CC of $O(n^{1.5})$ for a class of graphs that include bit-reversal.

In light of these results, for any size $n \in \mathbb{N}$ we construct a DAG of in-degree 2 and show (via a novel and somewhat involved analysis) that its CC is $\tilde{\Omega}(n^2)$. In particular for any constant $\epsilon > 0$ the CC grows faster than $\Omega(n^{2-\epsilon})$. The construction consists of $O(\log(n))$ layers of depth-robust graphs [EGS75,MMV13] which are graphs that still contain a long path even after some constant fraction of nodes is removed. Edges are then added to connect all layers into a single path spanning the entire graph. Finally, the nodes of neighbouring layers are connected using a new bit-mixing structure which, intuitively, ensures that if a set of nodes are close in one layer then their bit-mixed neighbours are widely dispersed across the next layer. The proof proceeds in three steps. The first (requiring most of the work) constructs a graph family with logarithmic in-degree and only for a subset of all sizes but with high CC. The second and third steps remove these relaxations. At its core, the proof (of step 1) eventually boils down to a case distinction reflecting an intuitive choice available to any pebbling of the graph. On the one hand, few pebbles could be used but (hopefully) resulting in many steps. Alternatively, much fewer steps may be needed but at the cost of using more pebbles. Of course an complete pebbling may alternate between or even mix these strategies at differ-

[8] For example when analysing bounded fan-in circuits, register allocation for programs with instruction sets having a limited number of arguments, or turing machines with a bounded number of tapes.

ent times and/or different parts of the graph so formalizing this intuition requires defining some finely tuned properties of a pebbling and careful analysis.

An Application to Memory-Hard Functions As an application of these tools we use the pebbling paradigm to define and construct a new MHF. In particular we put forth the first (formal) notion of a memory hard functions which enjoy parametrized *amortized* hardness in a parallel setting. For this we use the *parallel random oracle model* (pROM) in which algorithms can make batches of oracle queries in a single step. To motivate the new notion we show how it can be used to estimate the dollar cost of brute-forcing such a function either by building a custom circuit (e.g. an FPGA or ASIC) or by renting computational resources for the task. We provide a construction in the pROM of an MHF which can be computed sequentially with both work and memory at most n. In contrast, even when attempting to leverage parallelism and repeated evaluations of the MHF, the AT-complexity of a brute-forcing circuit still grows in $\tilde{\Omega}(n^2)$ per evaluation.[9] Due to our graph construction above having constant in-degree, we need assume only an ideal compression function rather than an arbitrary input-length RO. An added practical benefit of our construction (compared to that of [Per09]) is that the memory access pattern when sequentially evaluating the MHF (as done by say the login server) is independent of the input to $g_{n,i}$ which greatly reduces (if not eliminates) the potential for mounting successful cache-timing attacks [BM06].

The security proof makes use of the pebbling paradigm with the tools described above. Using the notion of hash graphs [DNW05,DKW11] we show how to obtain a family of functions f_G from (single source and sink) DAGs G in the pROM. The main technical contribution of this section is a theorem lower-bounding the amortized hardness f_G in the pROM using the CC of G in the parallel pebbling game. The reduction is given in terms of exact security and elucidates the effect of choices such as RO output length, the hardness parameter of the MHF and the success probability of an algorithm at computing the function. In contrast to past results on (sequential) amortized hardness [DGN03,DNW05,FLW13] it also makes precise the effect of finding collisions in the RO as the number of copies being computed grows.

Incidentally another consequence of this theorem and the algorithm for pebbling bit-reversal graphs is to give rise to a new parallel brute-force attack against the MHF of [FLW13] for one of the suggested practical parameter settings. In particular the new attack reduces the AT-complexity of special purpose brute-forcing hardware to $O(n^{3/2})$ compared to the cost of $O(n^2)$ in when brute-forcing on a sequential machine.

1.2 Related Work

The black pebbling game has a rich history and a full exposition of its application is beyond the scope of this work. Some notable examples though are its use in modelling register allocation [Set75], turing machine resources [Coo73,HPV77] and flowcharts [HP70,Pip80]. Moreover it has been used to explore space/time trade-offs for many important algorithmic tasks such as matrix multiplication [Tom78], the FFT [SS78,Tom78], integer multiplication [SS79a] and solving linear recursions [Cha73,SS79b]. More recently in the field of cryptography (a two colour variant of) the game has been used to prove lower-bounds on the number of cache misses [DNW05] or space required [DFKP13,ABFG13,FLW13] to compute certain functions by a sequential random access machine in the ROM. Finally an application of similar flavour demonstrated in [DKW11] shows how to ensure a function can be computed no more than once on memory-restricted secure hardware.

Another line of work uses novel pebbling games to study various classic parallel complexity classes. One important example is the two-player game of Dymond and Tompa [DT85] whose round complexity

[9] Technically these bounds are obtained for a fixed RO input size, success probability and a reasonable upper-bound on the number of RO calls performed by the circuit.

models runtime of an alternating turing machine or, equivalently [Ruz79], circuit depth. A variation of that game [VT89] was also used to characterize two parallel complexity classes; notably AC^1. Raz and KcKenzie [RM99] used a different two-player pebbling game to separate the monotone NC hierarchy. Quite recently, in [Cha13] it was shown that for any DAG with a single sink the minimal runtimes of [DT85] and [RM99] (as well as the S-complexity of a variant of the black pebble game used to study reversible computation [Ben89]) are all equivalent.

For the (sequential) black pebbling game several hard-to-pebble graphs have been explored. In [LT82] it was shown that bit-reversal graphs of size n have an ST-cost of $\Omega(n^2)$ which is optimal for graphs of equal size. Further they show that a graph consisting of a stack of superconcentrators not only has similar ST-cost but also exhibits an extreme space/time trade-off. That is if s pebbles are used then the time required grows exponentially in $n - s$. In [PTC76] a family of graphs is given that have S-cost $\Theta(n/\log(n))$ which is optimal for any graph of equal size [HPV77].

We use the depth-robust graphs construction of [MMV13] which is based on that of [EGS75]. While in the past they have been used to lower-bound circuit complexity and turing machine time [EGS75,Val77,Sch83], more recently they have been used in a positive context in the constructions of [MMV13,DFKP13].

Memory-hard functions were first introduced in by Percival in [Per09]. Although they have been well received in the security community they have received far less attention in the cryptographic community, with the notable exception of [FLW13] which focuses on the sequential case and provides security proofs using the pebbling paradigm based on ST-complexity in the black pebbling game.

Proofs-of-work have found a wide range of application such as countering spam email [DN93], web-site metering [FM98], countering denial-of-service attacks [JB99,Ada02] and many more [JJ99]. Especially round efficient variants have recently enjoyed an explosion of interest in the security community due to their use in maintaining a secure fully decentralized public transaction log. In this form they are being used to maintain various electronic cash systems [Nak09,Cha11], a distributed micro-blogging network [Mig13], a private messaging system [Nam11], and secure domain name system [BDLHA13] for example.

Another body of work, motivated by combating DoS attacks, focuses on proofs-of-sequential-work, known as *client puzzles*. These are 2-party protocols which aim to capture the intuition that the prover must perform a certain number of sequential computations (even given parallel computational resources) in order to convince the verifier. Most constructions [RSW96,Kv10,TBFN07,JM11] rely on assumptions about the parallel hardness of non-standard structured problems with the exception of [MMV11,MMV13] which are information theoretically secure in the random oracle model. Another common trait of these works is that they only exhibit a linear gap in the runtime of the (honest) prover and verifier except for that of [MMV13] which enjoys a polylogarithmic gap.

1.3 Overview

In Section 2 we fix our notation. In Section 3 we introduce the parallel pebbling game and the cumulative complexity measure. In Section 4 we show some lower-bounds and upper-bounds on the CC of various types of graphs including for the new family of high CC graphs. Turning to the main application, in Section 5 we introduce the pROM and a notion of amortized memory hardness for functions over strings. Finally in Section 6 we give the reduction lower-bounding the hardness of our memory-hard functions in terms of the CC of their underlying DAG. The appendix contains missing technical lemmata.

2 Tools and Notation

We denote by $\mathbb{N}$ the set of non-negative integers and for some condition C we write $\mathbb{N}_C$ to denote the subset of $\mathbb{N}$ satisfying C. For example $\mathbb{N}_{\leq 3} = \{0, 1, 2, 3\}$. For integers $a \geq b$ we write $[a] := \{1, \ldots, a\}$ and $[a, b] := \{a, a+1, \ldots, b\}$. For a set $\mathbb{H}$ we write $\mathcal{H} \leftarrow \mathbb{H}$ to denote sampling a fixed uniform random value $\mathcal{H}$ from $\mathbb{H}$.

The *source* nodes of a directed acyclic graph (DAG) are the nodes with in-degree 0. Similarly the *sink* nodes with out-degree 0. The *size* of a DAG is the number of nodes and its *depth* is the length of its longest path. We call a DAG *simple* if it has a single source and sink and in-degree 2. If for $i \in [2]$, $G_i = (V_i, E_i)$ are a pair of node disjunct DAGs then write $G_1 + G_2 := (V_1 \cup V_2, E_1 \cup E_2)$ to denote the DAG obtained by combining the two into a single graph. In particular graphs can be self-composed. For DAG $G = (V, E)$ and $m \in \mathbb{N}$ we write $G^{\times m}$ to denote the DAG obtained by viewing m independent copies of G as a single graph. In other words $G^{\times m}$ has $m * |V|$ nodes partitioned into m subsets eache with the same edge structure as G.

We use the following lemma bounding the success probability of a predictor getting short correlated hints. A proof can be found in [DKW11] for example.

Lemma 1. *Let $B = b_1, \ldots, b_u$ be a sequence of random bits. Let $\mathcal{P}$ be a randomized procedure which gets a hint $h \in \mathtt{H}$, and can adaptively query any of the bits of B by submitting an index i and receiving b_i. At the end of the execution $\mathcal{P}$ outputs a subset $S \subseteq \{1, \ldots, u\}$ of $|S| = k$ indices which were not previously queried, along with guesses for all of the bits $\{b_i | i \in S\}$. Then the probability (over the choice of B and randomness of $\mathcal{P}$) that there exits some $h \in \mathtt{H}$ for which $\mathcal{P}(h)$ outputs* all correct guesses *is at most $\frac{|\mathtt{H}|}{2^k}$.*

3 Parallel Graph Pebbling

We formalize an intuitive computational model of parallel graph pebbling and motivate a new complexity notion for graphs in this model.

Put simply we define a variant of the black pebbling game where pebbles can be placed according to the usual rules but in batches of moves performed in parallel rather than one at a time sequentially.

Definition 1 (Pebbling a Graph). *Let $G = (V, E)$ be a DAG and $T, S \subseteq V$ be node sets. Then a* (legal) pebbling of G (with starting configuration S and target T) *is a sequence $P = (P_0, \ldots, P_t)$ of subsets of V such that:*

1. $P_0 \subseteq S$.
2. *Pebbles are added only when their predecessors already have a pebble at the end of the previous step.*

$$\forall i \in [t] \quad \forall (x, y) \in E \quad \forall y \in P_i \setminus P_{i-1} \qquad x \in P_{i-1}.$$

3. *At some point every target node is pebbled (though not necessarily simultaneously).*

$$\forall x \in T \quad \exists z \leq t \qquad x \in P_z.$$

We call a pebbling of G complete *if $S = \emptyset$ and T is the set of sink nodes of G.*

In particular pebbles can be placed on a source node or removed from any node at any time.

We can now ready to define a our complexity notions for DAGs.

Definition 2 (Cumulative Pebbling Complexity). *Let G be a DAG, $P = (P_0, \ldots, P_t)$ be an arbitrary pebbling of G and Π be the set of all complete pebblings of G. Then the* (cumulative) cost *of P and then* cumulative complexity *(CC) of G are defined respectively to be:*

$$\mathsf{p\text{-}cost}(P) := \sum_{i=0}^{t} |P_i| \qquad\qquad \mathsf{cc}(G) := \min\{\mathsf{p\text{-}cost}(P) : P \in \Pi\}.$$

Moreover the space complexity *(SC) and* space/time complexity *(STC) are defined as follows:*

$$\mathsf{s\text{-}cost}(P) := \max\{P_i : i \in \{0, \ldots, t\}\} \qquad \mathsf{sc}(G) := \min\{\mathsf{s\text{-}cost}(P) : P \in \Pi\}$$

$$\mathsf{st\text{-}cost}(P) := k * \max\{P_i : i \in \{0, \ldots, t\}\} \qquad \mathsf{stc}(G) := \min\{\mathsf{st\text{-}cost}(P) : P \in \Pi\}$$

Amortized Graph Complexity. Unlike the standard pebbling complexity notions of SC and STC, for the parallel pebbling game, CC is also an amortized complexity notion. In fact it scales additively in the CC of the individual disconnected components of the graph. We prove this property of CC formally in the following lemma.

Lemma 2. *For $i \in [2]$ let $G_i = (V_i, E_i)$ be a pair of node disjunct DAGs and let $G = G_1 + G_2$. Then* $\mathsf{cc}(G) = \mathsf{cc}(G_1) + \mathsf{cc}(G_2)$.

In particular for any DAG G and $m \in \mathbb{N}$ it holds that $\mathsf{cc}(G^{\times m}) = m * \mathsf{cc}(G)$.

Proof. By concatenating optimal pebbling of the two sub-graphs (and removing any pebbles immediately from fully pebbled components) we obtain a pebbling P_{opt} of G which has a cost of $\mathsf{p\text{-}cost}(P_{opt}) = \mathsf{cc}(G_1) + \mathsf{cc}(G_2)$ which implies that $\mathsf{cc}(G) \leq \mathsf{cc}(G_1) + \mathsf{cc}(G_2)$.

Moreover any pebbling P of G can be split into a pair of pebbling P_1 and P_2 for the sub-graphs such that $\mathsf{p\text{-}cost}(P) = \mathsf{p\text{-}cost}(P_1) + \mathsf{p\text{-}cost}(P_2)$. For example to create P_1 all nodes from G_2 are removed from the component sets of P and any resulting component sets are completely removed. It is easy to verify that the result is a complete pebbling for P_1 according to Definition 1. Thus we also have $\mathsf{cc}(G) \geq \mathsf{cc}(G_1) + \mathsf{cc}(G_2)$. □

Trivially no version of Lemma 2 holds for SC. Indeed, by sequentially pebbling the individual components of a graph we see that the resulting SC of the graph is no more then the largest SC of any single component. Next, we also give a DAG for which the STC of pebbling a single copy is essentially the same as pebbling many copies.

Lemma 3. *Let m be a positive integer. There exists a DAG G with $\Theta(m^2)$ nodes and integer $n = \Theta(m)$ such that* $\mathsf{stc}(G^{\times n}) = O(\mathsf{stc}(G))$.

The proof consists mainly of a straightforward adaptation of an argument given in [Sav97] concerning pyramid graphs. Graph G is constructed out of two components, the first is a pyramid graph which is shown to require m pebbles but using only $\sqrt{m}$ steps while the second, a path of length m requires only 1 pebble but m steps. Thus while pebbling the chain of the first copy of G up to $\sqrt{m}$ other copies of G can be pebbled in parallel while only $2m$ pebbles are required in total. Moreover, with this pipelining approach the total time required only doubled compared to that needed for a single copy of G. The details follow.

Proof. Let $\bar{m} = m(m+1)/2$. The $\bar{m}$-node pyramid graph $P(\bar{m}) = (V_{\bar{m}}, E_{\bar{m}})$ has m layers of nodes where the layer $i \in [m]$ contains i nodes. We identify a node with a pair of integers (a, b) where $a \in [m]$ denotes its layer and $b \in [a]$ denotes its position in that layer. Thus $V_{\bar{m}} := \{(a, b) : a \in [m], b \in [a]\}$. Moreover node has at most two outgoing edges connecting it to the node in the next smallest layer at the same position (if it exists) and the previous position (if it exists). In symbols $E_{\bar{m}} := \{((a, b), (c, d) : a \in \{2, \ldots, m\},\ b \in [a],\ c = a - 1,\ d \in \{b - 1, b\} \setminus \{0, c + 1\}\}$. Let G be the graph consisting of $P(\bar{m})$ extended with $\bar{m}$-node chain $C(\bar{m})$ (i.e. a path of length $\bar{m}$) beginning at its peak, namely at node $(1, 1)$. In particular G has $2\bar{m} = O(m^2)$ nodes.

We first show that $\mathsf{stc}(G) = O(m^3)$. Since G contains (several) paths of length $\bar{m} + m$ any complete pebbling of G must have at least as many steps by the pigeonhole principle. Moreover in the standard (i.e. non-parallel) black pebbling game for graph $P(\bar{m})$ any complete pebbling must have a step P_i with at least m pebbles (see Lemma 10.2.1 of [Sav97] for a proof). The same argument caries over to our setting. In particular consider a step where $(1, 1)$ has a pebble on it. Then in that step trivially any path from nodes in layer m to $(1, 1)$ has a pebble on it. Thus there must be a first step P_i in the pebbling when this holds for all such paths. Moreover it must be that P_i contains (at least one) pebble on a node (m', j) for some $j \in [m']$ which is not in P_{i-1} as otherwise P_{i-1} would also have pebbles on all such paths. Let π a the newly covered path from (m, j) to $(1, 1)$ not already covered in P_{i-1}. Moreover for $a \in [m]$ let π_a be the path from (m, j) to $(1, 1)$ which merges with π as early as possible for any path connecting (m, j) and $(1, 1)$. (In particular $\pi_j = \pi$.) Then by assumption each π_a contains a pebble in P_i but the only pebble on π is on node (m, j). However all π_a are pairwise node disjunct except for the nodes they share with π. Thus there must be at least $m - 1$ further pebbles in P_i to cover each of these paths implying that $|P_i| \geq m$. Finally by pebbling each layer in parallel one step at a time followed by the nodes of $C(\bar{m})$ in sequence we obtain a complete pebbling of G with the largest set being $|P_1| = m$ and so $\mathsf{stc}(G) = m^2 + m\bar{m} = O(m^3)$.

Let G' consist of m independent copies of G. It remains to show that $\mathsf{stc}(G') = O(\mathsf{stc}(G))$ which we do by giving a simple algorithm for pebbling G'. We pebble the first copy of G layer by layer in parallel as just described. After each m steps we begin to simultaneously pebble a new copy of G in the same way. Thus at no point is more than one copy of $P(\bar{m})$ being pebbled and no more than m copies of G being pebbled which means that no step requires more than $2m$ pebbles. Moreover after m^2 steps we start pebbling the final copy of G and so G' is fully pebbled in $m^2 + m + \bar{m}$ steps. Thus $\mathsf{stc}(G') \leq 2(m^3 + m^2 + m\bar{m})$ which means $\mathsf{stc}(G') = O(m^3)$ as desired. □

We observe, informally, that CC also behaves relatively well for sequentially composed graphs. That is if G_1 has a single sink and G_2 has a single source and G is the graph obtained by connecting the sink of G_1 to the source of G_2 then $\mathsf{cc}(G) \leq \mathsf{cc}(G_1) + 2 * \mathsf{cc}(G_2)$. The result is obtained as follows. To pebble G first pebble G_1 optimally until a pebble is placed on it's sink. Then, without removing that pebble, use an optimal strategy to pebble G_2. The cost of the CC of second phase can be at most twice $\mathsf{cc}(G_2)$ implying the upper-bound.

4 The Complexity of Graphs

We show some lower and upper-bounds for the CC of some interesting families of DAGs. By pebbling the nodes of a graph one at a time in lexicographic order without ever removing a pebble it is clear that any DAG with n nodes has CC at most $n(n + 1)/2 = O(n^2)$. In the first part of this section we show that several potential candidates for achieving this bound actually fall well short of it. In the second part we give a construction which has CC of $\tilde{\Omega}(n^{2-\delta})$. As mentioned in the introduction we are especially interested in graphs with constant (or at least bounded) in-degree.

4.1 Upper-bounds on Cumulative Complexity of Graphs

It is easy to see that any DAG of size n and depth d has CC at most dn. In particular by pebbling all nodes possible during each iteration (and never removing a pebble) after any $d' \leq d$ number of iterations all nodes with distance at most d' from any source will have been pebbled. Moreover during any iteration at most n pebbles can ever be on G.

Lemma 4. *Let G be a DAG of size n and depth d. Then $\mathsf{cc}(G) \leq dn$.*

Superconentrators. One might hope that graphs which exhibit extreme trade-offs between the number of pebbles used and the time required in the sequential setting also exhibit high CC in the parallel setting. Unfortunately this is not, in general, the case.

A superconcentrator is a certain type of densely connected DAG.[10] In [LT82] it was shown that for any size n and number of pebbles $S \leq n$ there exists a DAG H_n of size n consisting of a certain stack of $O(\log(n))$ sequentially connected superconcentrators such that H_n can be sequentially pebbled in time T only if:

$$T = S\Omega\left(\frac{n}{S}\right)^{\Omega(n/S)}.$$

In particular if $S = o(n \log\log n / \log n)$ then time T becomes superpolynomial. However, in terms of achieving high parallel CC, intuitively the problem with H_n is that it is not deep enough. The construction of [LT82] uses Pippenger's construction of superconcentrators [Pip77] which, for any size $m \in \mathbb{N}$ results in depth only $O(\log(m))$. Thus H_n has depth only $\log^2(n)$. But then Lemma 4 implies that it can has parallel CC at most $O(n \log^2(n))$ which is well bellow our trivial upper-bound of $O(n^2)$.

Bit-Reversal Graphs. Another interesting family of graphs are bit-reversal graphs. It was shown in [LT82] that (in the sequential setting) any pebbling using S pebbles requires time T such that $ST = O(n^2)$. Again one might hope that such graphs could have high parallel CC. Instead we now describe an algorithm which can (in particular) pebble the bit-reversal graph of size n using cumulative cost of at most $O(n^{1.5})$.

Let $n \in \mathbb{N}$ be even. A *sandwich graph* is a chain of n nodes (numbered 1 through n) with arbitrary aditional edges connecting nodes from the first half of the chain with nodes of the second half of the chain such that no node has in-degree greater than 2. As a special case we get the permutation graphs and bit-reversal graphs of [LT82].

Lemma 5. *Any sandwich graph G of size n has $\mathsf{cc}(G) = O(n^{1.5})$.*

Proof. To avoid clutter we assume that $\sqrt{n}$ is an integer. The following ideas can easily be generalized. To pebble a sandwich graph execute the following rules for each iteration $i \in \mathbb{N}$.

1. If $i \mod \sqrt{n} = 0$ then place a pebble on node 1.
2. For each pebble on a node $v \in [n]$ place a pebble on node $v + 1$.
3. Remove any pebble on nodes $\{(n/2)+1, \ldots, n\}$ except the one on the highest valued node (if it exists).
4. Let m be the highest valued node with a pebble on it. Remove any pebble on nodes $v \in [n/2]$ except if $(i - v) \mod \sqrt{n} = 0$ or if there is an edge $(v, m + j)$ for some $0 < j < \sqrt{n}$ and $m + j > n/2$.

[10] The exact definition is not important here so we have omitted it.

We argue that the resulting pebbling is legal and complete, taking time n. The only violation of the pebbling rules can occur due to the second rule. It can inductively seen this never happens and moreover that at the end of iteration i there is a pebble on node i. The first $n/2$ steps are clearly legal and at the end there is a pebble on node $n/2$ and 1. Thus, in step $i = (n/2)+1$ a pebble is placed on node i legally. Suppose no violation has occurred up to iteration i and that a pebble is on node $i-1$. Then this is the only pebble on the second half of the chain since they were removed in iteration $i-1$ due to the fourth rule. Suppose there is no edge of the form (v, i) for some $v \in [n/2]$. Then applying the second rule in iteration i is legal and results in a pebble on i so we are done. Suppose such an edge does exist. Since each $\sqrt{n}$ iterations a fresh pebble begins walking along the chain until it reaches node $n/2$ one of those pebbles has been on v within the last $\sqrt{n}$ iterations. Thus the fourth rule prevents it from being removed during the previous $\sqrt{n}$ iterations and so at the end of iteration $i-1$ there is still a pebble on v making the placement of a pebble on i legal.

It remains to show that the end of no iteration are there more than $2\sqrt{n}+1$ pebbles on G. The third rule implies that at most one pebble can ever be on the second half of G. At most $\sqrt{n}$ nodes will remain due to the first clause of the fourth rule. Moreover since the nodes of G have in-degree at most 2 there can be at most $\sqrt{n}$ edges from the first half of G ending in nodes of the second half between m and $m+\sqrt{n}$. Thus at most as many will be spared removal due to the second clause of the fourth rule.

Taken together shows that the cost of the resulting pebbling can be at most $n*(2\sqrt{n}+1) = O(n^{1.5})$. □

We remark that a consequence of Lemma 5 is that the conjecture of [FLW13], lower-bounding the ST-cost of their memory-bound function does not hold given parallelism. In that work the authors restrict themselves to a sequential setting, but given the suggested application of their MHF as a "password scrambler" for storing passwords and the highly parallel environments used to brute-force these, the parallel version of the conjecture is still of interest. In more detail the authors construct a graph function from stack of $\lambda \in \mathbb{N}_{\geq 1}$ bit-reversal graphs. They suggest as practical the values of $\lambda \in \{1, 2, 3, 4\}$. However for $\lambda = 1$ their construction is a sandwich graph and so Lemma 5 shows that it can be brute-forced with say an ASIC having significantly lower AT-complexity then could be hoped for given their results for the sequential computational settings.

4.2 Lower-bounds on Cumulative Complexity of Graphs

A brief warm up for a naïve attempt at building a graph with high CC can be found in Appendix A. The remainder of this section is focused on proving the following theorem.

Theorem 1. *For any $n \in \mathbb{N}$ large enough there is a simple (efficiently and explictely constructable) DAG G_n such that:*

$$\mathsf{cc}(G_n) \in \Omega\left(\frac{n^2}{(\log^{10} n)(\log\log n)^2}\right).$$

On the highest level the proof can be broken down into the following steps.

1. Prove a relaxation of the theorem. The graphs may have in-degree up to $\log^3 n$. Moreover they need exist only for a (not too) sparse sub-sequence of possible sizes $n \in \mathbb{N}$.
2. Remove the relaxation on the in-degree and number of sinks.
3. Fill in graphs for the missing sizes to obtain a sequence covering every (large enough) size $n \in \mathbb{N}$.

Proving the Relaxed Case. Our goal in this part of the proof to show the following relaxation of Theorem 1. The first difference is that instead of constant in-degree it requires only that the in-degree be polylogarithmic in the size. Later in step (2), we show how to reduce the in-degree δ of any DAG such that the effective result is a DAG whose CC is a δ^{-3} multiple of the original. The second difference is that we don't need a graph for every possible size. Finally we need the edge count to stay low because when reducing in-degree to constant many edges will be replaced by intermediary nodes and we want to ensure that the sizes of the resulting graph sequence are not to sparse so that we can finally build a sequence containing a graph of every size.

Lemma 6 (Poly-logarithmic In-degree). *There exists an infinite sequence of (efficiently and explictely constructable) DAGs $(H_1, H_2, \ldots)$ with sizes $n_1, n_2, \ldots$, in-degrees $\delta_1, \delta_2, \ldots$ and a single source and sink such that for all $i \in \mathbb{N}$:*

1. $n_i \leq (2 + \log i)2^i$
2. $\delta_i \leq 3\log^3(n_i) + 1$
3. $\mathsf{cc}(H_i) \in \Omega\left(\frac{n_i^2}{\log(n_i)(\log\log n_i)^2}\right)$

To prove the lemma we give a construction of a graph $G_{\omega,\phi,i}$ with three parameters. Then we show how they affect the in-degree and complexity of the graph which we summarized in the following lemma. It represents the technical core of the proof of the theorem.

Lemma 7. *Fix any $\phi \in \mathbb{N}$, $0 < \omega \leq 4^{-\phi}/6$.*

- *Let $C_{\omega,\phi}$ be a constant such that some set $\{D_{4^{\phi-j}\omega,2^i} : j \in \mathbb{N}_\phi\}$ of depth-robust graphs exists $\forall i \geq C_{\omega,\phi}$.*
- *Let C_ϕ be a constant such that $\forall i \geq C_\phi$ we have $\log(i)/i \geq 2^{-\phi}/3$.*

If both constants exist then $\forall i \geq \max\{C_{\omega,\phi}, C_\phi, 11\}$ the DAG $G_{\omega,\phi,i}$ from Construction 2 has a single source and sink, size n_i and:

1. $n_i = (\phi + 1)(2^i)$.
2. $\mathsf{indeg}(G_{\omega,\phi,i}) \leq 3i^3 + 1$.
3. $\mathsf{cc}(G_{\omega,\phi,i}) \geq \frac{\omega^2}{144} i^3 (2^i)^{2-2^{-\phi}}$.

Before we get into the construction and its proof we first show that Lemma 7 does indeed suffices for Lemma 6. For this we use the following lemma which is a trivial reformulation of the Lemma 4.4 from [MMV13].

Lemma 8 ([MMV13], Lemma 4.4). *There exists a constant C such that for any $i \geq C$ there exists an (efficient and explicitely constructable) graph D_{2^i} of size 2^i which is ω-depth robust for all $\omega \in (0, 1/6)$.*

Proof (Proof of Lemma 6). We define ω and ϕ as functions of i and show that the resulting sequence of graphs exists, is efficiently and explictely constructable and has that it the three properties of Lemma 6.

Set $\phi_i := \lceil \log i \rceil$ and $\omega_i := 4^{-\phi_i}/6$. We first argue that there exists an universal constant $i_0 \geq 11$ such that $\forall i \geq i_0$ it holds both C_{ω_i,ϕ_i} and C_{ϕ_i} exist and that $i \geq \max\{C_{w_i,p_i}, C_{\phi_i}\}$. In particular the graph $G_{\omega_i,\phi_i,i}$ from Lemma 7 exists. Recall that $I := 2^i$. We use the construction from the proof of Lemma 8 (found in [MMV13]). Thus, there is a (small) universal constant C such that for all $i \geq C$ the exists an efficient explicit construction of a graph – call it D_I – such that $\mathsf{indeg}(D_2^i) < i^3$ and moreover, D_I is ω-depth robust for all $\omega \in (0, 1/6)$.[11] In other words, for large enough i we can always set $C_{\omega_i,\phi_i} = C$.

[11] We remark that improving the bound on the in-degree of D_I to say $i^{3-\gamma}$ results in multiplying $\mathsf{cc}(G_n)$ by $\log^{3\gamma}(n)$ in Theorem 1. For example the in-degree of [MMV13] can actually be bounded by $\tilde{\mathcal{O}}(i^2)$ rather then i^3 as we have used here. Using that tighter bound would increase $\mathsf{cc}(G_n)$ by a multiplicative factor of almost $\log^3(n)$. However for ease of exposition we have left out this more fine-grained analysis.

We also observe that the inequality $\log(i)/i \geq 2^{-\lceil \log i \rceil}/3$ is satisfied by say any $i \geq 2$. Thus we can set $i_0 = \max C, 11$ to conclude that for all $i \geq i_0$ the graph $G_{\omega_i,\phi_i,i}$ of Lemma 7 exists.

Now for each $i \in \mathbb{N}$ we define $H_i := G_{\omega_{i_0+i},\phi_{i_0+i},i_0+i}$. Clearly, since the depth-robust graphs we use are efficiently and explictely constructable so is H_i.

It remains to show that H_i has the three properties required by Lemma 6. Fix any $i \in \mathbb{N}$. To avoid clutter, we write $j = i_0 + i$ and omit the subscripts for ϕ and ω. The first property follows directly from the definition of ϕ_i. For the second property we see that:

$$\delta_i = \mathsf{indeg}(G_{\omega,\phi,j}) \leq 3j^3 + 1 = 3\log^3(2^j) + 1 \leq 3\log^3((\lceil \log j \rceil + 1)2^j)) + 1 = 3\log^3(n_i) + 1.$$

Let $n := (\phi + 1)2^j$ be the size of H_j. For the third property we have:

$$\begin{aligned}
\mathsf{cc}(H_i) = \mathsf{cc}(G_{\omega,\phi,j}) &= \frac{\omega^2}{144} j^3 (2^j)^{2-2^{-\phi}} \geq \frac{\omega^2 j^3 (2^j)^2}{144(2^j)^{2^{-\lceil \log j \rceil}}} \geq \frac{\omega^2 j^3 2^{2j}}{288} \\
&= \frac{(4^{-\lceil \log j \rceil})^2 j^3 2^{j^2}}{10368} \geq \frac{2^{j^2}}{10368 j} \geq \frac{(\lceil \log j \rceil + 1)^2 2^{j^2}}{10368(\lceil \log j \rceil + 1)^2 j} \\
&\geq \frac{n^2}{10368 j (2 + \log j)^2} \geq \frac{n^2}{10368 \log(n)(2 + \log\log n)^2} \\
&\in \Omega\left(\frac{n^2}{\log n (\log\log n)^2}\right)
\end{aligned}$$

□

We now turn to proving the main technical lemma. We begin with the construction of $G_{\omega,\phi,i}$. Let $I := 2^i$ and $\omega \in (0, 1)$. A graph D of size I is said to be *ω-depth robust* if after removing any node subset of size less than ωI their remains some path of length at least $(1 - 2\omega)I$. In the following we denote by $D_{\omega,I}$ a ω-depth robust graph of size I with $\mathsf{indeg}(D_{\omega,I}) = i^3$ (for example the ones constructed explicitely in [MMV13]).

We also need the *bit-mixing* function $\mathfrak{b}_{i,\phi}(x, s)$. Its role is to produce an outputs consisting of a concatenation of the lower order bits of x and s such that for all pairs of inputs sharing the high order bits of x the images are as widely dispersed as possible. However its definition is somewhat technical and requires several constants which we will repeatedly use through out the proof. For an implicit but fixed i and ϕ we define the following three constants:

$$\alpha := \left\lfloor (1 - 2^{-\phi})i \right\rfloor \qquad \beta := \left\lfloor (1 - 2^{-\phi})i + 3\log i \right\rfloor \qquad R := 2^{\beta - \alpha}$$

Notice that for any i such that $(\log i)/i \leq 2^{-\phi}/3$ it holds that $\beta \leq i$. By assumption we have restricted ourselves to the setting $i \geq C_\phi$ so we can conclude that $\beta \leq i$.

For implicit but fixed i and ϕ we define the *bit-mixing* function $\mathfrak{b}(x, s) : N_{<2^i} \times N_{<R} \to N_{<2^i}$ to be $\mathfrak{b}_{i,\phi}(x, s) = (2^{i-\alpha}x + 2^{i-\beta}s) \mod 2^i$. In particular, for any distinct inputs $(x_1, s_1) \neq (x_2, s_2)$ where x_1 and x_2 share the $i - \alpha$ most significant bits we have $|\mathfrak{b}(x_1, s_1) - \mathfrak{b}(x_2, s_2)| \geq 2^{i-\beta}$.

Construction 2 *For $\phi, i \in \mathbb{N}$, $I := 2^i$ and $0 < \omega \leq 4^{-\phi}/6$ we construct $G_{\omega,\phi,i}$ as follows:*

1. *We begin with $\phi + 1$ layers where layer $j \in \mathbb{N}_{\leq \phi}$ is a copy of $D_{4^{\phi-j}\omega,I}$. In particular $G_{\omega,\phi,i}$ has $(\phi+1)I$ nodes which we identify with the set $V = \mathbb{N}_{<(\phi+1)I}$ such that the nodes of $D_{4^{\phi-j}\omega,I}$ are numbered in topological order. When convenient we also use the notation $\langle a, b\rangle$ to denote the node $Ia + b$.*
2. *We augment $G_{\omega,\phi,i}$ with any missing edges on the path $\{(j, j + 1) : j \in [(\phi + 1)I]\}$.*
3. *We connect the nodes from one layer to their bit-mixed analogues in the next layer. More precisely we add the edges $\{(\langle l - 1, \mathfrak{b}(j, s)\rangle, \langle l, j\rangle) : l \in [\phi + 1], j \in \mathbb{N}_{<I}, s \in \mathbb{N}_{<R}\}$.*
4. *At the first layer we added the edges $\{(v, u) : u \in \mathbb{N}_{<I}, v \in \{u - 2 - \min\{u - 2, i^3\}, \ldots, u - 2\}\}$.*

Intuition. Before we dive into the details we first take a moment to provide some intution for the rest of this proof. To show that the construction has the properties we need we first fix some legal pebbling of $G_{\omega,\phi,i}$. Intuitively the idea show that making progress at layer j requires one of two strategies. At the moment one starts to pebble layer j either many pebbles lie on layer $j-1$ or few do.

Many Initial Pebbles: For this case, a large pebbling cost is incurred because many of the initial pebbles (or decedents thereof) must be kept on the graph until most of layer j has been completed. To show we define a particular notion of dependence between pebbles. Intuitively these capture a sequence of pebbles lying on layer $j-1$ which tie some initial pebble to a pebble on the origin of a particular despairer edge leading to a node v in layer j. We call such a dependency chain a *track for node v and dispersal edge e* and we say it is *complete (at time t)* if a pebble is placed at origin of e for the first time at time t. First we show that many tracks are actually pebble disjoint. Next we fix any pebbling step P_t just before some node v in layer j is pebbled. We observe that only tracks for node v can become complete at time t. That is all tracks for nodes further along in layer j are not complete yet. But there are at least as many pebbles still on layer $j-1$ at time t as there are pebble disjunct incomplete tracks. A careful formalization of these concepts allows us to lower-bound just how many such tracks remain at any time t. Thus summing over the time steps it takes to pebble layer j we obtain a lower-bound incurred pebbling cost.

Few Initial Pebbles: Intuitively, for this case we obtain a large pebbling cost because most of layer $j-1$ must be repebbled to make progress on layer j. In particular, the depth-robustness of layer $j-1$ ensures in that, having only a few initial pebbles on layer $j-1$, there is a long path somewhere in that layer with no initial pebbles on it. Yet because of the dispersal edges originating from it going to layer j, it must be fully pebbled by the time layer j is complete. In other words such a pebbling of layer j contains an (almost) complete pebbling of the previous layer. We use this property to prove the statement inductively on the number of layers in the graph.

One caveat with the above reasoning is that a pebbling of layer j may actually switch between the two types of strategies, sometimes using few pebbles layer $j-1$ and sometimes using many. Therefor in the proof we first divide up layer j into groups of nodes located contiguously on a path. On the one hand, the groups must be pebbled in sequence as they lie on a single path and so we can sum their individual pebbling costs to obtain a lower-bound on the pebbling cost of the entire layer. On the other hand, they are short enough that any pebbling must really use one of the two strategies while pebbling the entire group. Thus by the above reasoning each one of these groups will require a large pebbling cost in it's own right.

We are now ready to formally show that the construction satisfies the conditions of Lemma 7.

Proof. Each layer of $G_{\omega,\phi,i}$ is acyclic and edges between layers only go from lower layer to a higher one so the DAG is acyclic. Since $G_{\omega,\phi,i}$ includes a chain spanning all nodes it has a single source and sink. Moreover the DAG consists of $(\phi+1)$ copies of depth-robust graphs, each with I nodes which implies property 1.

Properties 2 is also straightforward. Nodes at any layer $j \in [\phi]$ have at most $R+1 \leq 2i^3+1$ more incoming edges (added in steps 2 and 3 of Construction 2) then they inherited from $D_{4^{\phi-j}\omega,I}$, which has in-degree i^3, giving a total of $3i^3+1$. Nodes at layer 0 have at most i^3+1 more incoming edges (added in steps 2 and 4) then they inherited from $D_{4^{\phi}\omega,I}$ giving a total of $2i^3+1$. So for $i>0$ property 2 holds.

This leaves the proof of property 3 which we do by induction on ϕ. We actually prove a stronger statement which lends itself well to an inductive proof and implies property 3 as a special case. We denote the nodes at layer $j \in [\phi+1]$ as $V_j := \{\langle j, v\rangle \in V : v \in \mathbb{N}_{<I}\}$.

Claim. For any $G_{\omega,\phi,i}$ assume some set of less than ωI nodes $U \subseteq V_\phi$ (and incident edges) are removed from V_ϕ. Then pebbling the remainder of V_ϕ using any starting configuration having no pebbles on V_ϕ has a CC of at least $\frac{\omega^2}{144} i^3 (I)^{2-2^{-\phi}}$.

The special case when no pebbles are removed and the starting configuration is empty implies the lemma.

Case $\phi = 0$**:** Fix any $0 < \omega \leq 1/6$. Recall that $G_{\omega,0,i}$ consists of a single layer. The ω-depth-robustness of D_I implies that after removing less than ωI nodes there remains a path $\Psi = (v_1, \ldots, v_z)$ of length (at least) $z \geq (1 - 2\omega)I$ at this layer. Given that the starting configuration is empty the nodes along this path must receive their first pebble at strictly increasing time intervals. To bound $\mathsf{cc}(G_{\omega,0,i})$ we sum the pebbles of the graph at the time steps just before each node Ψ received it's first pebble. That is we bound $\mathsf{cc}(G_{\omega,0,i})$ by Σ, the sum of the in-degrees of the nodes in Ψ (since placing a pebble on a node requires having a pebble on all it's parents at the end of the previous step).
Assume for a moment no parents of nodes in Ψ where removed. Then each node v_j would have in-degree at least $\min\{v_j - 1, i^3\}$. Moreover, the smallest possible value of Σ would be obtained when $v_{i^3+1} = i^3$ in which case $\Sigma = i^3 * (I(1 - 2\omega)) - i^6/2$. But no node can be a parent of more than i^3 of the nodes on the path, so even after removal Σ can decrease by at most $i^3(\omega I)$. So we can write:

$$\mathsf{cc}(G_{\omega,0,i}) \geq i^3 * (I(1 - 2\omega)) - i^6/2 - i^3 I\omega = i^3 I \left(1 - 3\omega - i^3/2I\right) \geq \frac{\omega^2}{4} i^3 I.$$

where the last inequality holds for any $\omega \leq 1/6$ and $i \geq 11$.

Case $\phi > 0$**:** Fix any $0 < \omega \leq 4^{-\phi}/6$. As in the base case, we focus on the long remaining path Ψ in layer ϕ. We split Ψ into subgroups such that the cost of pebbling the path is at least the sum of the costs of pebbling each group. Next we show that many of these groups must have a considerable length and, for those that are long enough, they must be relatively expensive to pebble as they either require almost completely re-pebbling layer $\phi - 1$ or they require maintaining a large number of pebbles on that layer.
The ω-depth-robustness of $D_{\omega,I}$ implies that after removing less than ωI nodes there remains a path Ψ of length $z = (1 - 2\omega)I$ at this layer. To make optimal use of the bit-mixing sub-structure we select all nodes in a group to share their $i - \alpha$ most significant bits.[12] Furthermore, for simplicity we only consider large enough groups as only these contribute significantly to the final pebbling cost.
More precisely for $a \in \mathbb{N}_{<i-\alpha}$ we define *group* $\Gamma_a := \{(\phi, v) \in p : \left\lfloor \frac{v}{2^\alpha} \right\rfloor = a\}$ while a *heavy* group Γ_a is one where $|\Gamma_a| \geq (1 - 3\omega)2^\alpha$. We show that there are a large number of such groups.

Claim. There are at least $2^{i-\alpha}/3$ heavy groups.

Proof. Suppose this is not the case. A heavy group has size at most 2^α and the rest have size less than $(1 - 3\omega)2^\alpha$. Thus Ψ has length z which is at most:

$$\begin{aligned} z &< (2^{i-\alpha}/3)2^\alpha + (2^{i-\alpha} - 2^{i-\alpha}/3)(1 - 3\omega)2^\alpha \\ &= 2^i/3 + (2/3)2^i(1 - 2\omega - \omega) \\ &= 2^i/3 + (2/3)2^i(1 - 2\omega) - (2/3)2^i\omega \\ &= (2/3)2^i(1 - 2\omega) + (1/3)2^i(1 - 2\omega) \\ &= z \end{aligned}$$

which is a contradiction. □

[12] Intuitively this ensures that all incoming edges from the previous layer have pairwise widely disperesed origins.

Suppose the node set $U \subset V_\phi$ has been removed from $G_{\omega,\phi,i} = (V, E)$ and fix an optimal complete pebbling $P = (P_0, \ldots, P_t)$ with target $T = V_\phi \setminus U$ and starting configuration S such that $S \cap V_j = \emptyset$. Since our long path $\Psi \subseteq T$ but it is not contained in S it must be fully pebbled by P. We divide the execution of P into time periods. For each group Γ_a the corresponding period Π_a consists of all steps after Γ_a has received its first pebble but before Γ_{a+1} has received its first pebble. More precisely for $a \in \mathbb{N}_{<i-\alpha}$ the *period* Π_a the set of intervals:

$$\Pi_a := \{j \in \mathbb{N}_{\leq t} : \exists j_0 \leq j, P_{j_0} \cap \Gamma_a \neq \emptyset \ \ \wedge \ \ \forall j_1 \leq j, \forall b > a, P_{j_1} \cap \Gamma_b = \emptyset\}.$$

and by convention we define $\Pi_{-1} := S$. Period Π_a is said to *start* at $\mathsf{ps}_a := \min\{\Pi_a\}$ and *end* at $\mathsf{pe}_a := \max\{\Pi_a\}$. To estimate the cost incurred during each period we define its associated *cost set* $CP_a := \{\langle j, s\rangle : j \in \Pi_a, s \in P_j\}$ which corresponds to the set of pebbles on the graph during interval Π_a. Thus the *group cost* of pebbling Γ_a is $|CP_a|$. More generally, since the periods are pairwise disjunct but jointly cover all time intervals we can write:

$$CC(G_{\omega,\phi,i}) = \mathsf{p\text{-}cost}(P) = \sum_{j=1}^{t} |P_j| = \sum_{a \in \mathbb{N}_{<2^{i-\alpha}}} |CP_a|.$$

Now if we can show that for any heavy group Γ_a we can bound $|CP_a| \geq (\frac{\omega^2}{24})2^{\alpha+\beta}$ then we are done because then we can write:

$$\begin{aligned} CC(G_{\omega,\phi,i}) &\geq (2^{i-\alpha}/3)(\omega^2/24)2^{\alpha+\beta} = \frac{\omega^2}{72}2^{i+\beta} \\ &= \frac{\omega^2}{72}2^{i+\lfloor(1-2^{-\phi})i+3\log(i)\rfloor} \\ &\geq \frac{\omega^2}{144}i^3 2^{i+i(1-2^{-\phi})} \\ &= \frac{\omega^2}{144}i^3(2^i)^{2-2^{-\phi}}. \end{aligned}$$

To show this bound on $|CP_a|$ we will show that pebbling a heavy period requires either re-pebbling almost all the lower layers or else requires keeping many pebbles on the them. In either case the resulting cost will be large.

For a given node, we define the notion of an m-*track* which is a sequence of pebbles placed during Π_a that were needed to eventually place a pebble on x by traversing the m^{th} bit-mixed parent $\langle \phi - 1, b(x, m)\rangle$. Moreover the m-*generator* of x is the node closest to node $\langle \phi - 1, \mathfrak{b}(x, m)\rangle$ which has a pebble on it at the beginning of any m-track of x.

Definition 3 (Track and Generator). *For $m \in \mathbb{N}_{<R}$ an* m-track *of node $x = \langle \phi, v\rangle \in \Gamma_a$ is a function $r : \{-1, \ldots, n_x - 1\} \to V \setminus U$ with* length n_x *such that:*

1. *Interval $\mathsf{ps}_a + n_x$ is the first time x is pebbled. That is $\mathsf{ps}_a + n_x = \min\{l : x \in P_l\}$.*
2. *Either $r(j) = 0$ or a pebble is on node $r(j)$ at time $\mathsf{ps}_a + j$. That is $\forall j$ we have $r(j) \in P_{\mathsf{ps}_a+j} \cup \{0\}$.*
3. *The track ends on the source of the m^{th} bit-mixed edge. Formally $r(n_x - 1) = \langle \phi - 1, \mathfrak{b}(v, m)\rangle$.*
4. *The remainder of the track follows the path $0, 1, 2, \ldots$ through the graph. Formally $\forall j < n_x - 2$ either $r(j + 1) = r(j)$ or $r(j + 1) = r(j) + 1$.*

A maximal m-*track of node x has maximal $r(-1)$ which is called the* m-generator $g_{x,m}$ *of x. That is if $\mathbb{T}$ is the set of tracks for x the generator $g_{x,m}$ of x is:*

$$g_{x,m} := r(-1) \quad \textit{for a maximal track} \quad r \in \operatorname*{argmax}_{r \in \mathbb{T}}\{r(-1)\}.$$

The m-generator of a node is well defined. In particular we prove the following claim.

Claim. For any group Γ_a, node $x \in \Gamma_a$ and $m \in \mathbb{N}_{<R}$ there is (at least one) m-track r.

Proof. As $x = \langle \phi, v \rangle \in \Gamma_a \subseteq T$ is a target it was pebbled by P; in particular for the first time during period Π_a at time $\mathsf{ps}_a + n_x$ as dictated by property 1. We define $r(n_x - 1) = \langle \phi - 1, b(v, m) \rangle$ satisfying property 3. For all other values we set $r(j-1) = r(j)$ if either there is a pebble on $r(j)$ at time $\mathsf{ps}_a + j - 1$ or $\rho(j) = 0$. Otherwise we set $r(j-1) = r(j) - 1$. Clearly property 4 holds.
To see property 2 suppose that it didn't hold. Then for some j we have $r(j) \notin P_{\mathsf{ps}_a + j} \cup \{0\}$. Fix the largest such j. Since property 1 holds $j < n_x - 1$. Moreover, by definition $r(j) = r(j+1) - 1$ and there was a pebble on $r(j+1)$ at time $\mathsf{ps}_a + j + 1$ but not at time $\mathsf{ps}_a + j$. But since the pebbling P is legal and a pebble was placed on node $r(j+1)$ at time $\mathsf{ps}_a + j + 1$ there must have been a pebble its predecessor $r(j+1) - 1$ at in the previous interval $\mathsf{ps}_a + j$ which is a contradiction to our definition of j. □

Recall that the proof is complete if we can show the following claim.

Claim. For any heavy group with group cost CP_a it holds that $|CP_a| \geq \frac{\omega^2}{24} 2^{\alpha+\beta}$.

We need a step where not too many pebbles are on the graph. Since Γ_a is heavy it has at least $(1-3\omega)2^\alpha$ nodes arranged along a path and so they must be pebbled one after another. Therefor the period lasts for at least $|\Pi_a| \geq (1 - 3\omega)2^\alpha \geq (\omega/3)2^\alpha$ where the second inequality follows from the fact that $\omega < 4^{-\phi}/3 < 1/6 < 3/10$. Fix a $\mathsf{ps}_a \leq q < \mathsf{ps}_a + (\omega/3)2^\alpha$ for which $|P_q| < (\omega/3)2^i$. If none exists we are done since we can write:

$$|CP_a| = \sum_{j \in [(\omega/3)2^\alpha]} \left|P_{\mathsf{ps}_a + j - 1}\right| \geq (\omega/3)2^\alpha * (\omega/3)2^i \geq (\omega/3)^2 2^{\alpha+\beta} \geq \frac{\omega^2}{24} 2^{\alpha+\beta}.$$

We make a case distinction on the size of $k \in [|\Gamma_a| * R]$; the number of distinct generators for all nodes in Γ_a and all $m \in \mathbb{N}_{<R}$.

Case $k < 2^\beta \omega/3$**:** For this case we show that almost the entire sub-graph $G_{4^{\phi-1}\omega,\phi-1,i}$ needs to be re-pebbled. For this we want to use our inductive assumption by showing that P contains a complete pebbling for all but $(4\omega)I$ nodes of $V_{\phi-1}$. More specifically, after all nodes $\overline{U}$ are removed there remains a complete pebbling $\overline{P}$ (contained in P) of target nodes $\overline{T}$ from starting configuration $\overline{S}$ where:

$$\begin{aligned}
F &:= (P_{q+1} \cup \ldots \cup P_{\mathsf{pe}_a}) \cap V_{\phi-1} \\
\overline{U} &:= (V_{\phi-1} \setminus F) \cup (P_q \cap V_{\phi-1}) \\
\overline{T} &:= F \setminus \overline{U} \\
\overline{S} &:= P_q \setminus (V_\phi \cup V_{\phi-1}) \\
\overline{P} &:= (\overline{S}, P_{q+1} \setminus (\overline{U} \cup V_\phi), \ldots, P_{\mathsf{pe}_a} \setminus (\overline{U} \cup V_\phi))
\end{aligned}$$

It is easy to verify that all of $\overline{T}$ is pebbled by $\overline{P}$ and $\overline{S}$ contains only nodes pebbled at time P_q. Moreover after removing nodes and edges the constraints on when a pebble can be placed legally can only become easier to satisfy.[13] Therefor $\overline{P}$ is in fact a legal and thus complete pebbling of $\overline{T}$ with starting configuration $\overline{S}$.

[13] For example in the case of $P_{q+1} \setminus \overline{U}$ the only question of legality could be about pebbles placed somewhere in $V_{\phi+1}$. But either their parents are in a lower layer (in which case they were pebbled $P_q \setminus \overline{U}$) or they are in $\overline{U}$ and so were removed from the graph. Thus all remaining parents had a pebble at time q and so the placement at time $q+1$ was legal. Continuing this argument for the remaining iterations shows that $\overline{P}$ is legal.

Still, to use induction to prove this case we need more; namely that $\overline{U}$ is not too large. Trivially $\overline{T}$ contains only nodes from layer $\phi - 1$ while $\overline{S}$ contains no nodes from that layer. So it remains to show that $\left|\overline{U}\right| < (4\omega)I$. For this we first lower-bound the size of F. As k is small we know that some generator is shared by many nodes $\Phi_g \subseteq V_\phi$ in the group. Intuitively this means that much of $V_{\phi-1}$ was repebbled in order to reach all of Φ_g starting from generator g. More formally we show that for all $v \in \Phi_g$ (but one) a large chunk of nodes immediatly preceding v's bit-mixed predecessors in $V_{\phi-1}$ are in F. Since bit-mixing achieves a guaranteed minimum separation between the origins of any edges incoming to Φ_g this means that F must be relatively large.

Claim. Let $m \in \mathbb{N}_{<R}$, $v_x = \langle \phi, x \rangle \in \Gamma_a$ and g be its m-generator. We call the pair of nodes v_x, m *distant* if there exists an $\bar{m} \in \mathbb{N}_{<R}$ and $v_y = \langle \phi, y \rangle \in \Gamma_a$ with $\bar{m}$-generator g such that $\mathfrak{b}(y, \bar{m}) < \mathfrak{b}(x, m)$. For any distant pair of nodes v_x, m it holds that:

$$f_{x,m} := \{\phi - 1\} \times \{\mathfrak{b}(x, m) - 2^{i-\beta} + 1, \ldots, \mathfrak{b}(x, m)\} \subseteq F.$$

Proof. Let $p_{y,\bar{m}} := \langle \phi - 1, \mathfrak{b}(y, \bar{m}) \rangle$ and $p_{x,m} := \langle \phi - 1, \mathfrak{b}(v_x, m) \rangle$. The image $\mathtt{Im}(r)$ of an m-maximal track r of v_x is an interval in $[g, p_{x,m}] \subset \mathbb{N}$ and moreover $p_{y,\bar{m}} \in [g, p_{x,m}]$. As v_x and v_y are in the same group they share their $i - \alpha$ most significant bits. Therefor by definition of the bit-mixing function $p_{x,m} - p_{y,\bar{m}} \geq 2^{i-\beta}$ and so $[p_{x,m} - 2^{i-\beta} + 1, p_{x,m}] \in \mathtt{Im}(r)$ where the 1 is added since F might not contain all of P_{ps_a} but $g \in P_{\mathsf{ps}_a}$. □

For any distinct pair of distant nodes $(x, m) \neq (y, \bar{m})$ we have that $f_{x,m} \cap f_{y,\bar{m}} = \emptyset$. For any generator g there is a unique non-distant pair (x, m) having g as a generator. Moreover during the first $(q - \mathsf{ps}_a)$ steps of the period at most $(q - \mathsf{ps}_a)$ nodes of the group can be pebbled. Thus at step q we are left at with at least $(|\Gamma_a| - (q - \mathsf{ps}_a))$ nodes to pebble, each of which has R maximal tracks. At most k of these are not distant and so we can write:

$$\begin{aligned}
|F| &\geq ((|\Gamma_a| - q + \mathsf{ps}_a)R - k)2^{i-\beta} \\
&\geq (((1 - 3\omega)2^\alpha - (\omega/3)2^\alpha)2^{\beta-\alpha} - (\omega/3)2^\beta)2^{i-\beta} \\
&= \left(1 - \frac{11\omega}{3}\right) 2^i
\end{aligned}$$

Recall that by assumption $|P_q| < (\omega/3)2^i$. The we can write

$$\left|\overline{U}\right| < 2^i - \left(1 - \frac{11\omega}{3}\right) 2^i + (\omega/3)2^i = (4\omega)2^i.$$

Since each set in $\overline{P}$ is a subset of the corresponding subset of P it follows by induction on sub-graph $G_{4w,\phi-1,i}$ that

$$|CP_a| \geq \mathsf{p\text{-}cost}(\overline{P}) \geq \frac{(4\omega)^2}{288} i^3 (2^i)^{2-2^{-\phi+1}} \geq \frac{w^2}{18} 2^{2(3/2 \log i + i(1-2^{-\phi}))} \geq \frac{\omega^2}{18} 2^{\alpha+\beta}.$$

Case $k \geq 2^\beta \omega/3$: In this case we have many generators. Intuitively we'd like to conclude that for much of the period many pebbles must lie on the graph. To do this we show that no pair of maximal tracks for distinct generators ever interesect. That is they never share a pebble. Moreover, by definition, as long as long as the last node of a track hasn't recieved a pebble that track must still contain at least one pebble. Finally we observe the, as all nodes in the group lie on the path Ψ, not too many tracks leading to bit-mixed predecessors of Ψ can be completed during any given step of the

pebbling. Together these facts imply that during much of the begining of the pebbling there are many unfinished tracks, each with it's own pebble on the graph. The details follow.

To formalize this intuition we first take care of the corner case when $k \leq 3$. That would mean that $2^\beta \leq 9/\omega$. Since the pebbles of (heavy) group Γ_a must be pebbled in sequence (as they lie on a chain) we have that:

$$|CP_a| \geq |\Gamma_a| \geq (1-3\omega)2^\alpha \geq \frac{\omega(1-3\omega)}{9}2^{\alpha+\beta} \geq \frac{\omega^2}{18}2^{\alpha+\beta}$$

where the last inequality holds for any $\omega \leq 1/6$.

Suppose now that $k > 4$. Let $v_0 \leq v_1 \leq \ldots \leq v_{k-1}$ be a set of (not necessarily distinct) nodes in Γ_a and $\{m_i \in \mathbb{N}_{<R} : i \in \mathbb{N}_{<k}\}$ such that for all $i \neq j \in \mathbb{N}_{<k}$ the m_i-generator of v_i is distinct from the m_j-generator of v_j. All nodes v_i lie on the same path and equality can hold for at most R consecutive nodes. Recall that n_{v_j} is such that $\mathsf{ps}_a + n_{v_j}$ is the first step in which v_j receives a pebble. Then $n_{v_j} \geq n_{v_{j-1}}$ where equality holds only if $v_j = v_{j-1}$. So $n_{v_j} \geq j/R$. Recall that unless a track has reached node 0 (which can happen to tracks only with generator 0) there is, by Definition 3, always a pebble on the track. We claim that no such pebble can be shared between tracks originating from distinct generators.

Claim. For any pair of nodes $x, y \in \Gamma_a$ and integers $m, \bar{m} \in \mathbb{N}_{<R}$ let r_x be a maximal m-track of x and r_y be a maximal $\bar{m}$-track of y. If $g_{x,m} < g_{y,\bar{m}}$ then there exists no time j where $r_x(j) = r_y(j)$.

Proof. Suppose the claim is false and that for some j we have $r_x(j) = r_y(j)$. Then we can construct an m-track $\overline{r}$ for x contradicting the maximality of r_x. Essentially $\overline{r}$ follows r_x down to time j and then follows r_y until time -1. Formally set $\overline{r}(z) := r_x(z)$ for any $j \leq z < n_x$ and $\overline{r}(z) := r_y(x)$ for any $-1 \leq z < j$. Clearly $\overline{r}$ is a track for x since r_x was a track for x and property 2 held for r_y as well. But it also holds that $r_x(-1) = g_x < g_y = \overline{r}(-1)$ which is a contradiction to the maximality of r_x. □

In summary each v_j adds at least n_{v_j} to the total cost of pebbling the group except maybe for the one whose generator is 0. This implies that:

$$|CP_a| \geq \sum_{j \in [k-1]} n_{v_j} \geq \frac{(k-1)k}{2R} \geq \frac{3k^2}{8R} \geq \frac{\omega^2}{24}2^{\alpha+\beta}$$

where the third inequality follows since $k > 4$.

□

Constant In-degree We show how to reduce the in-degree of a graph at the cost of some of its CC.

Lemma 9 (Reducing In-degree). *Let H be an (efficiently and explictely constructable) DAG of size $n \in \mathbb{N}$ with one source and sink and in-degree δ. Then there exists a simple (efficiently and explictely constructable) DAG J of size at least n and at most δn such that $\mathsf{cc}(H) \leq (\delta - 1) * \mathsf{cc}(J)$.*

Proof. To build J from H we replace each node v of H with in-degree δ_v by a path of δ_v nodes each receiving one of v's incoming edges. Clearly if H is efficiently and explictely constructable then so is J. To bound $\mathsf{cc}(J)$ we convert any pebbling P of J into one for H and bound its cost in terms $\mathsf{p\text{-}cost}(P)$.

For node v we write $\delta_v := \mathsf{indeg}(v)$. Fix a topological ordering O of $H = (V, E)$, let $s \in V$ be the (unique) source node of H and for $v \in V$ let $p_{(v,1)} < p_{(v,2)} < \ldots < p_{(v,\delta_v)}$ be the parents of v sorted

according to O. Also by convention $p_{(s,0)} := \perp$. We identify each node of $J = (\overline{V}, \overline{E})$ with a pair of values $\overline{V} \subseteq V \times (V \cup \{\perp\})$. Initially $\overline{V} = \{(s, \perp)\}$ and $\overline{E} = \emptyset$. For each $v \in V \setminus \{s\}$ add the nodes $\{(v, p_{(v,1)}), \ldots, (v, p_{(v,\delta_v)})\}$ to $\overline{V}$ and edges $\{((v, u_i), (v, u_{i+1})) : i \in [\delta_v]\}$ to $\overline{E}$. For each edge $(v, w) \in E$ add the edge $((v, p_{(v,\delta_v)}), (w, v))$ to $\overline{E}$.

Since each $v \in V$ is replaced by at at least one and at most δ_v nodes in $\overline{V}$ the size of J is between n and δn. Let $P = (P_0, \ldots, P_t)$ be an optimal pebbling of J. We convert it into a pebbling $S = (S_0, \ldots, S_t)$ of H. For each P_i build S_i according to the following pair of rules.

1. For all $(v, p_{(v,\delta_v)}) \in P_i$ add v to S_i.
2. For all $(v, p_{(v,j)}) \in P_i$ with $v \neq s$ and $j < \delta_v$ add $\{p_{(v,1)}, \ldots, p_{(v,j)}\}$ to S_i.

We show that if P is complete for J then S must be complete for H. As P is complete, the sink (x, δ_x) of J is pebbled in P and so, by rule 1, x, the sink of H, must be pebbled in S. We verify that S is also legal, i.e. that it doesn't violate the rules of the pebbling game. Suppose it does. Let $i \in [t]$ be the first violation of the pebbling rules and $v \in S_i$ be a pebble violating the rules. Clearly $v \neq s$.

Suppose the pebble was added to S_i by applying rule 1. Then $(v, p_{(v,\delta_v)}) \in P_i$ and, since $v \notin S_{i-1}$ it must be that $(v, p_{(v,\delta_v)}) \notin P_{i-1}$. As P is legal, the parents of $(v, p_{(v,\delta_v)})$ must be in P_{i-1}. On the one hand, that means that $(p_{(v,\delta_v)}, \delta_{p_{(v,\delta_v)}}) \in P_{i-1}$ and so $p_{(v,\delta_v)} \in S_{i-1}$ by rule 1. On the other hand, if $\delta_v > 1$ then $(v, p_{(v,\delta_v-1)}) \in P_{i-1}$ and so all remaining parents of v are pebbled in S_{i-1} by rule 2. Thus $v \in S_i$ is a legal move.

Suppose instead that the pebbled was added to S_i by applying rule 2 to a pebble $(w, p_{(w,j)}) \in P_i$. In particular for some $1 \leq l \leq j$ we have $v = p_{(w,l)}$. Since $v \notin S_{i-1}$ it must be that $l = j$ and so $(p_{(w,j)}, \delta_{p_{(w,j)}}) = (v, \delta_v) \in P_{i-1}$. But then by rule 1 $v \in S_{i-1}$ which is a contradiction.

It remains only to observe that for each P_i there are at most $\delta - 1$ pebbles in S_i. So we get that:

$$\mathsf{cc}(H) \leq \mathsf{p\text{-}cost}(S) \leq (\delta - 1) * \mathsf{p\text{-}cost}(P) = (\delta - 1) * \mathsf{cc}(J).$$

□

A Graph for Every Size The last step in the proof of Theorem 1 is to fill in the sequence of graphs so that we have a graph for every possible size.

Recall that a simple DAG is one with a single source and sink and in-degree 2.

Lemma 10. *Let $n_0 < n_1 < n_2 < \ldots$ be an infinite sequence of increasing positive numbers such that for some constant $a \in \mathbb{R}$ and all $i \in \mathbb{N}$ it holds that $n_{i+1}/n_i \leq a$. Let $\chi : \mathbb{N} \to \mathbb{N}$ be a monotone increasing function such that there exists an infinite sequence of simple (efficiently and explictely constructable) DAGs $(J_1, J_2, \ldots)$ where J_i has size at most n_i and $\mathsf{cc}(J_i) = \Omega(\chi(n_i))$. Then there exists an infinite sequence of simple (efficiently and explictely constructable) DAGs $(G_1, G_2, \ldots)$ such that G_j has size j and $\mathsf{cc}(G_j) = \Omega(\chi(j/a))$.*

Proof. In a nutshell the proof idea is to fill in the gap between J_i and J_{i+1} by extending J_i with increasingly long paths attached to its sink node. As the difference in CC between J_i and and J_{i+1} is bounded and χ is monotone increasing we can show that we can find a constant to show that the asymptotic behavior $\mathsf{cc}(G_j)$ remains essentially unchanged compared to $\mathsf{cc}(J_i)$.

Let $\psi : \mathbb{N}_{\geq n_0} \to \mathbb{N}$ be the function such that $\psi(j) := \max\{i : n_i \leq j\}$. For $j \in \mathbb{N}_{<n_0}$ define G_j to be a path of length j which implies that it is a simple (efficiently and explictely constructable) DAG and has both size and CC j. For $j \in \mathbb{N}_{\geq n_0}$ define DAG G_j to consist of $J_{\psi(j)}$ with a path of length $j - n$ appended

to its sink, where $n \leq n_{\psi(j)}$ is the size of $J_{\psi(j)}$. Then G_j is simple (efficiently and explictely constructable) if $J_{\psi(j)}$ was and it has size $n + (j - n) = j$.

By assumption $\exists i_0 \in \mathbb{N}$ and constant $c \in \mathbb{R}$ such that $\forall i \geq i_0$ $\mathsf{cc}(J_i) \geq c * \chi(n_i)$. So if we set $j_0 = n_{i_0}$ then we see that:

$$\forall j > j_0 \quad \mathsf{cc}(G_j) = \mathsf{cc}(J_{\psi(j)}) + (j - n_i) \geq c * \chi(n_{\psi(j)}) \geq c * \chi(j/a).$$

□

Proof of Theorem 1 Now that we have all the pieces we can prove Theorem 1.

Proof. Let $(H_1, H_2, \ldots)$ be the sequence of (efficiently and explictely constructable) DAGs from Lemma 6 and for each $i > 0$ let H_i have size $n_i \leq (2 + \log i)2^i$ and in-degree $\delta_i \leq 3\log^3(n_i) + 1$. Then Lemma 9 implies the existence of a sequence of simple (efficiently and explictely constructable) DAGs $(J_1, J_2, \ldots)$ where $h_i \in [n_i, n_i\delta_i] \subseteq \mathbb{N}$ is the size of J_i and its CC is:

$$\begin{aligned}\mathsf{cc}(J_i) \geq (\delta_i - 1)^{-1} * \mathsf{cc}(H_i) \in (\delta_i - 1)^{-1} * \Omega\left(\frac{n_i^2}{\log(n_i)(\log\log n_i)^2}\right) &\subseteq \Omega\left(\frac{(n_i\delta_i)^2}{\delta_i^3\log(n_i)(\log\log n_i)^2}\right) \\ &\subseteq \Omega\left(\frac{(n_i\delta_i)^2}{\log^{10}(n_i)(\log\log n_i)^2}\right) \subseteq \Omega\left(\frac{h_i^2}{\log^{10}(h_i)(\log\log h_i)^2}\right).\end{aligned}$$

Now we want to fill in the DAGs for missing sizes using Lemma 10. Clearly $\mathsf{cc}(H_i)$ grows at least as fast as a monotonically increasing function in its size. Thus it remains only to show that the ratio of the sizes of consecutive graphs in the sequence is bounded by a constant.

Observe that for all $i \in \mathbb{N}_{\geq 1}$ it holds that $n_i \leq 2^{2i}$. So we set $s_i := 2^{2i}(3(2i)^3 + 1)$ in which case $h_i \leq n_i\delta_i \leq s_i$ and $s_1 < s_2 < \ldots$. Next we fix an arbitrary $i \in \mathbb{N}_{\geq 1}$ and write:

$$\frac{s_{i+1}}{s_i} \leq \frac{2^{2(i+1)}(3(2(i+1))^3 + 1)}{2^{2i}(3(2i)^3 + 1)} = 4 * \frac{24(i+1)^3 + 1}{24i^3 + 1} < 4 * \left(\frac{i+1}{i}\right)^3 \leq 32.$$

So by Lemma 10 there exists a sequence of simple (efficiently and explictely constructable) DAGs $(G_1, G_2, \ldots)$ such that G_j has size j and:

$$\mathsf{cc}(G_j) \in \Omega\left(\frac{(j/32)^2}{\log^{10}(j/32)(\log\log(j/32))^2}\right) \subseteq \Omega\left(\frac{j^2}{\log^{10}(j)(\log\log j)^2}\right).$$

□

5 Memory Harndess in the Parallel Random Oracle Model

We formally describe the *Parallel Random Oracle Model* (pROM) together with a notion of computational hardness lower-bounding the expected amortized cumulative memory usage required to compute a given function in the pROM. We motivate the definition by showing how it can be used to provide meaningful bounds on the cost building a circuit to repeatedly evaluate the function.

5.1 The Parallel Random Oracle Model

The primary computational entity in the pROM is a probabilistic algorithm[14] T which has access to (arbitrarily many) parallel copies of a (stateless) oracle $\mathcal{O}$. The algorithm is iteratively applied to a state which it updates after processing the current iteration's batch of oracle queries. We assume $\mathcal{O}$ is sampled uniformly from an oracle set $\mathbb{O}$ and that T may depend on $\mathbb{O}$ but not $\mathcal{O}$.

At the beginning of each iteration T receives input (x, σ_i) where $\sigma_1 = \emptyset$ and σ_i has bit-length $|\sigma_i|$ which we assume to be strictly greater than 0 for all but the first and last states.[15] Then T issues a batch $\mathbf{q}_i$ of queries to $\mathcal{O}$ containing $|\mathbf{q}_i|$ individual queries. Next it receives the responses and finally it outputs a new state σ_{i+1}.[16] At the end of any iteration T can append values to a special output register and it can end the computation by outputting a special terminate symbol $\perp$ on that register. When this happens the contents y of the output register (excluding $\perp$) is considered the output of the computation. To denote the process of sampling an output we write $y \leftarrow T^{\mathbb{O}}(x)$. We say that T *computes* $f_{\mathcal{O}}$ on input x with probability ϵ if $\Pr[y \leftarrow T^{\mathbb{O}}(x) : y = f_{\mathcal{O}}(x)] \geq \epsilon$ taken over the coins of T and the choice of a uniform random oracle $\mathcal{O} \leftarrow \mathbb{O}$. Algorithm T is said to "run in time $t \in \mathbb{N}$ and make at most q queries" if it appends $\perp$ to its output register in iteration $t + 1$ and $\sum_i |\mathbf{q}_i| \leq q$.

A *family of oracle functions (with domain D)* is a set $f = \{f_{\mathcal{O}} : D \to R\}_{\mathcal{O} \in \mathbb{O}}$; that is a set of (deterministic) functions with domain D indexed by oracles from $\mathbb{O}$. For $m \in \mathbb{N}$, the *m-composition* of f is the family of oracle functions $f^{\times m} = \{f_{\mathcal{O}}^{\times m} : \mathbf{x} \in D^m \mapsto (f_{\mathcal{O}}(x_1), \ldots, f_{\mathcal{O}}(x_n)) \in R^m\}$. We call an input to a composed function *valid* if its m components of x are pairwise distinct.

Definition 4 (Amortized Cost). *The* cost *of running algorithm T on input x is the expected cumulative sum of its memory usage.*

$$\mathsf{cost}_{\mathbb{O}}(T, x) := \mathop{\mathbb{E}}_{T,\mathbb{O}} \left[\sum_i |\sigma_i| \right]$$

where the expectation is over the coins of T and choice of a uniform random $\mathcal{O} \leftarrow \mathbb{O}$.

Moreover for any $q \in \mathbb{N}$ and $\epsilon \geq 0$ the complexity *of an oracle function f is the lowest cost for which some input can be evaluated correctly with at least probability ϵ. More precisely:*

$$\mathsf{comp}_{q,\epsilon}(f) := \min_{x,T} \{\mathsf{cost}_{\mathbb{O}}(T, x)\}$$

where the minimum is taken over all (valid) inputs x and all algorithms T making at most q queries and computing $f_{\mathcal{O}}(x)$ with probability at least ϵ.

Finally for $m \in \mathbb{N}$ the amortized complexity *of an oracle function f is:*

$$\mathsf{a\text{-}comp}_{m,q,\epsilon}(f) := \min_{\bar{m} \in [m]} \left\{ \frac{\mathsf{comp}_{q,\epsilon}(f^{\times \bar{m}})}{\bar{m}} \right\}.$$

Equipped with this notion of amortized hardness we can now define our (somewhat informal) notion of an MHF.

[14] The exact computational model (say turing machines or RAMs) is not important.

[15] That is we assume, without loss of generality, that all intermediary states contain information. This is because the results of any computation done before an empty intermediary state have no effect on the computation done afterwards and so could have been skipped at no cost to the algorithm's ability of computing an output.

[16] In contrast to [DKW11], we do not assume that state σ_i contains information about future random coins used by T. For example in the turing machine setting, σ_{i+1} does *not* include the contents of the random tape. Rather the random tape is sampled once and fixed at the beginning of the execution.

Definition 5 (Memory-Hard Function). *Let $\mathbb{H}$ be a family of oracles (all with the same fixed input size and output size). A* memory-hard function *(MHF) in the pROM with is a collection $f = \{f_n : n \in \mathbb{N}\}$ with* hardness parameter *n of funciton families $f_n = \{g_{n,\mathcal{H}} : \mathcal{H} \in \mathbb{H}\}$ such that:*

NON-TRIVIALITY**:** *There exists a sequential algorithm T (i.e. one making batches of oracle queries of size only 1) such that $\forall n, \mathcal{H}$ and inputs x the output of $T^{\mathcal{H}}(n, x)$ is $g_{n,\mathcal{H}}(x)$ and T runs in time at most n using memory at most n.*

MEMORY-HARDNESS**:** *For a reasonable upper-bounds on the number q of queries made and the number m of copies evaluated and lower-bound on the success probability ϵ relative to the random oracle output size it holds that $\mathsf{a\text{-}comp}_{m,q,\epsilon}(f) = \tilde{\Omega}(n^2)$.*

The precise definition of "reasonable" above is given in Corollary 1 bellow.

To motivate these definitions we briefly describe how they can be used to lower-bound the effective (dollar) cost of repeteadly computing an MHF using a custom circuit such as an ASIC or FPGA.[17]

Custom Circuits and AT-Complexity: Traditionally, in VLSI design, the effective cost of a circuit is calculated as the product of the area (i.e. number of gates) and the time taken by the circuit.[18] This effective cost is called the *AT-complexity* of a circuit [Tho79]. Denote by AT_a the smallest AT-complexity of any circuit computing a copies of f (with probability at least ϵ while making at most q queries). We argue that, for large enough m, the value $\mathsf{a\text{-}comp}_{m,q,\epsilon}(f)$ represents an powerful tool for bounding the AT_a *any* realistic number of copies of f. In particular it provides a good lower-bound on the amortized AT-complixity per evaluation of any circuit repeatedly computing f.

On the one hand $AT_a \geq \mathsf{comp}_{a,q,\epsilon}(f)$ since at least as many memory components must be on the chip as are needed per iteration on expectation. On the other hand, for any integers $m \geq b > 0$ it follows immediatly from the definition of CC that $\mathsf{comp}_{b,q,\epsilon}(F) \geq b * \mathsf{a\text{-}comp}_{m,q,\epsilon}(f)$. In other words, by lower-bounding $\mathsf{a\text{-}comp}_{m,q,\epsilon}(f)$ we immediately obtain a lower-bound on AT_a for any $a \leq m$.

6 From Graphs to Hard Functions in the pROM

We show how to derive a family of oracle functions in the pROM from a given DAG. Then we give a reduction lower-bounding the amortized cost of computing the a random function from the family by the cumulative complexity of the graph.

The following definition is essentially taken from [DKW11].

Definition 6 (Labeling). *Let $G = (V, E)$ be a DAG, L be an arbitrary* label set*, $\mathbb{H} = \{\mathcal{H} : V \times L^* \to L\}$ be the set of all such functions.*[19] *For function $\mathcal{H} \in \mathbb{H}$ label $\ell \in L$ the $(\mathcal{H}, \ell)$-*labeling *of G is a mapping $\mathsf{lab} : V \to L$ defined recursively by:*

$$\forall v \in V \;\; \mathsf{lab}(v) = \begin{cases} \mathcal{H}(v, \ell) & : \mathsf{indeg}(v) = 0 \\ \mathcal{H}(v, \mathsf{lab}(v_1), \ldots, \mathsf{lab}(v_z)) & : \mathsf{indeg}(v) > 0 \end{cases}$$

[17] We remark that in either case other costs may, of course, also be involved in the final tally. However we merely aim to lower-bound the costs, not fully characterize them.

[18] More generally if the circuit only computes the correct value with probability ϵ then the effective cost is also multiplied by ϵ.

[19] Technically this is an infinite set making a uniform selection of an oracle ill-defined. However it suffices to consider the finite subset of oracle which take up to the in-degree of G number of labels as input. However for the sake of exposition we will slightly abuse notation and terminology.

where $\{v_1, \ldots, v_z\}$ *are the parents of* v *arranged in some lexicographic order.*

The graph functions (of G and $\mathbb{H}$) *are the members of the family of oracle function* $f = f_{\mathbb{H}}^{G}$*, indexed by* $\mathbb{H}$*, mapping* L *to* L^z *where* z *is the number of sink nodes in* G*. For input* $\ell \in L$ *the value of* $f_{\mathcal{H}} \in f$ *is* $f_{\mathcal{H}}(\ell) = (\mathsf{lab}(v_1), \ldots, \mathsf{lab}(v_z))$ *where the* v_i *are the sinks of* G *arranged in lexicographic order and* lab *is the* $(\mathcal{H}, \ell)$*-labeling of* G*.*

6.1 The Cost of Collisions

Let $f = f_{\mathcal{H}}^{G}$ be some oracle graph. When computing it on $m > 1$ distinct inputs $\boldsymbol{x} = (x_1, \ldots, x_m)$ – i.e. when computing $f^{\times m}$ on a valid input – it may happen that we encounter collisions for $\mathcal{H}$ which could drastically reduce the cost of computing some of the instances. On the other hand when computing $f_{\mathcal{H}}^{G^{\times m}}(\boldsymbol{x})$ collisions can not help since each node of $v \in G^{\times m}$ is distinct and so the first component of the inputs to $\mathcal{H}$ defining any labels required to compute $f_{H}^{G^{\times m}}$ is unique. In the following lemma we bound how much can be saved using collisions by relating the cost of computing $f^{\times m}$ making optimal use of collisions to the cost of computing f independently some (smaller) number of times.

Lemma 11. *Let* $G = (V, E)$ *be a DAG with* $n = |V|$*,* L *be a label set of size* $w = \log(|L|)$*, and* $\mathbb{H}$ *be the associated oracle set. Then for any* $m \in \mathbb{N}$ *with* $\beta := \max\{0, 1 - mn2^{-(w+1)}\}$ *and for any* $q \in \mathbb{N}$ *and* $\epsilon \geq 0$ *it holds that:*

$$\mathsf{comp}_{q,\epsilon}(f_{\mathbb{H}}^{\times m}) \geq \mathsf{comp}_{q,\epsilon}(f_{\mathbb{H}}^{(G^{(\times \beta m)})}).$$

Proof. We assume that an algorithm computing $f^{\times m}$ can detect and make use of collisions for free. In particular when such a collision occur say between inputs x_1 and x_2 to f we assume that T can compute $f(x_2)$ for free.

We say that a pair of labelings of G *collide* if there exists a node $v \in V$ with the same label. Let lab_1 and lab_2 be a pair of labelings of G with the same oracle but distinct inputs ℓ_1 and ℓ_2. The the probability that they collide is $p_2 = 1 - (1 - 2^{-w})^n \leq n2^{-w}$ where the inequality holds for any n and w in $\mathbb{N}_{>0}$. Thus the expected number of collisions in m labelings for distinct inputs is $p_m = \frac{1}{2} * m(m-1)p_2 \leq m^2 n 2^{-(w+1)}$. In other words on expectation only $m - p_m \geq \beta * m$ copies of f need to be computed. □

Intuitively for a large enough range of the RO, collisions should be too rare to provide significant short cuts in any realistic computation (even under the generous assumptions we make about the algorithm in the proof). Indeed, suppose we want to ensure that at most a $\beta = 1 - 2^{-80}$ fraction of the expected cost can be saved by leveraging collisions. Then if the RO range size is $w = 160$ the above lemma shows that for any graph size n and up to $m \leq \frac{2^{81}}{n}$ evaluations of f must be done before more then a β fraction of the expected cost can be saved.

6.2 The Reduction

We can now state the main technical lemma which is at the heart of the reduction. It bounds the cost of computing a graph function *once* in terms of the CC of the underlying graph. Before we prove it, using ideas adapted from [DKW11,DNW05], we give some intuition explaining why the bound should hold. Then we state the reduction bounding the amortized cost of a graph function and use the lemma to prove it. We also state a corollary of the reduction which simplifies the bound under some realistic constraints on the size of the RO, the probability of succeeding and the amount of work done. Finally we spend the remainder of this section proving the technical lemma.

Lemma 12 (One-Off Cost). *For G, L with $w := |L|$, $\mathbb{H}$ and $f = f^G_{\mathbb{H}}$ as in Definition 6. Then for any $q \in \mathbb{N}_{>0}$ and $\epsilon \geq q2^{-w}$ it holds that:*

$$\mathsf{comp}_{q,\epsilon}(f) \geq \frac{\epsilon \mathsf{cc}(G)(w - \log q)}{\epsilon\overline{\epsilon} + 1}$$

where $\overline{\epsilon} := -\log(\epsilon - q2^{-w})$.

Intuition. Before moving on we first provide a bit of intuition behind the terms in the lemma. Let $c_t := \epsilon(\mathsf{cc}(G)(w - \log q) - t\overline{\epsilon})$. We claim that an execution running in time t must have cost greater than both t and c_t. Supposing we believe this. Since c_t is monotonically decreasing in t, the lowest bound is obtained for an execution running in time t_0 such that $t_0 = c_{t_0}$. Solving for t_0 we get exactly the bound from the lemma.

We provide some intuition for the claim. Since by assumption at least one bit is stored per iteration, the fact that the cost must be at least the run time is clear. The intuition why the cost must be at least c_t goes as follows. Initially one might hope for a lower-bound of $c_t = \mathsf{cc}(G)w$. To see why c_t must decrease in $\log q$ consider, for example, an algorithm which makes every possible query to $\mathcal{O}$ in one batch during the first iteration. It can immediately compute f for any input. The lower-bound must reflect this (and more refined approaches that explorer large parts of the oracles domain).

Next, the $t\overline{\epsilon}$ term is needed because an algorithm may try and save on some $\overline{\epsilon}$ bits preferring instead to guess them when they are needed. For an extreme example, suppose the optimal algorithm T of computing f requires t iterations such that the result Q of an initial oracle call is to be stored and reused as input to an oracle call during all subsequent $t - 1$ iterations. Consider an algorithm T' that computes f according to T except that it doesn't store the last $\overline{\epsilon}$ bits of Q and instead uses a fixed (hard-coded) guess for those bits each time it needs Q. With probability $2^{-\overline{\epsilon}}$ (over the choice of $\mathcal{H}$) the guess will be correct and T' will compute f correctly having saved $t\overline{\epsilon}$ bits in cost compared to T. The above lemma shows the limits of how much can be saved with this approach in terms of the success probability ϵ and any choice of q and w.

Finally the multiplicative ϵ term is needed to account for algorithms which fail cheaply. Continuing with the above example, suppose after initially obtaining oracle response Q algorithm T' would continue the computation as we described only if its hard-coded guess for the last $\overline{\epsilon}$ bits of Q are correct and otherwise it would immediately terminate at no further cost. Then with probability (almost) ϵ it would compute f correctly having saved $t\overline{\epsilon}$ compared to the cost of T and with probability $(1 - \epsilon)$ it would fail, but with no further cost, essentially matching our lower-bound for the same time t and query count q of algorithm T and any choice of success probability ϵ.

Amortized Cost. Before we prove the Lemma 12 state the main result of this section which reduces the *amortized* complexity of a graph function to the CC of the associated graph.

Theorem 3 (Amortized Cost). *Let G be a DAG with $n := |V|$, L be a label set of size $w := \log(|L|)$, $\mathbb{H} = \{\mathcal{H} : V \times L^* \to L\}$ be the set of all such functions and $f = f^G_{\mathbb{H}}$ be the associated graph functions. Then for any $q \in \mathbb{N}_{>0}$, $m \in \mathbb{N}$ and $\epsilon > q2^{-w}$ it holds that:*

$$\mathsf{a\text{-}comp}_{m,q,\epsilon}(f) \geq \frac{\beta\epsilon \mathsf{cc}(G)(w - \log q)}{\epsilon\overline{\epsilon} + 1}$$

where $\overline{\epsilon} := -\log(\epsilon - q2^{-w})$ and $\beta := \max\{0, 1 - mn2^{-(w+1)}\}$.

Proof. For $\bar{m} \in [m]$ let $\beta_{\bar{m}} := \max\{0, 1 - mn2^{-(w+1)}\}$. The theorem follows from the following calculation.

$$\begin{aligned}
\mathsf{a\text{-}comp}_{m,q,\epsilon}(f) &= \min_{\bar{m}\in[m]} \left\{ \frac{\mathsf{comp}_{q,\epsilon}(f^{\times \bar{m}})}{\bar{m}} \right\} \\
&\geq \min_{\bar{m}\in[m]} \left\{ \frac{\mathsf{comp}_{q,\epsilon}\left(f_{\mathbb{H}}^{(G^{(\times \beta_{\bar{m}}\bar{m})})}\right)}{\bar{m}} \right\} \\
&\geq \min_{\bar{m}\in[m]} \left\{ \frac{\epsilon \mathsf{cc}(G^{\times \beta_{\bar{m}}\bar{m}})(w - \log q)}{\bar{m}(\epsilon\overline{\epsilon} + 1)} \right\} \\
&= \min_{\bar{m}\in[m]} \left\{ \frac{\beta_{\bar{m}} \epsilon \mathsf{cc}(G)(w - \log q)}{\epsilon\overline{\epsilon} + 1} \right\} \quad = \frac{\beta_m \epsilon \mathsf{cc}(G)(w-\log q)}{\epsilon\overline{\epsilon}+1}
\end{aligned}$$

where the second, third and fourth (in)equalities follow from Lemma 11, Lemma 12 and Lemma 2 respectively and the last equality holds since β_m is monotonically decreasing in m.

Before we prove Lemma 12 we first formulate a corollary of the theorem which shows that for realistic settings of the variables the intuition that the cost should be at least $\epsilon w \mathsf{cc}(G)$ is essentially correct. In particular we only consider algorithms which can make up to the birthday bound number of RO calls.

Corollary 1. *Let G, n, w, m, $q > 0$, ϵ and f be as in Theorem 3 with the added constraint that the RO range size w is large enough that $w > 13$ and:*

- *Not too many copies of f are computed. In particular $mn \leq 2^{w-2}$.*
- *Not too many oracle queries are made. In particular $q \leq 2^{w/2}$.*
- *The probability of computing f is reasonably large. In particular $\epsilon \geq 2^{-w/2}$.*

Then it holds that:

$$\mathsf{a\text{-}comp}_{m,q,\epsilon} \geq \frac{\epsilon w \mathsf{cc}(G)}{4}.$$

Proof. To prove the claim we must show that $\frac{\beta(w-\log q)}{\epsilon\overline{\epsilon}+1} \geq \frac{w}{4}$. Notice that we can bound β by:

$$\beta \geq 1 - mn2^{-(w+1)} \geq 1 - 1/8 = 7/8.$$

Moreover since by assumption $q \leq 2^{w/2}$ it must be that $\frac{w-\log q}{w} \geq 1/2$. Thus it suffices to prove that $(\epsilon\overline{\epsilon} + 1)^{-1} \geq 4/7$ which, after substituting in the definition of $\overline{\epsilon}$ and moving terms around, is equivalent to the inequality $\epsilon - 2^{\frac{-3}{4\epsilon}} \geq q2^{-w}$. Suppose that $\epsilon \leq \frac{4}{3w}$ then we can write $\epsilon - 2^{\frac{-3}{4\epsilon}} \geq \epsilon - 2^{-w} \geq q2^{-w}$. Suppose instead that $\epsilon > \frac{4}{3w}$. Since $w > 8$ the interval $I = (\frac{4}{3w}, 1/10]$ is not empty. Suppose then that $\epsilon \in I$ and define the function $g(\epsilon) := \epsilon - 2^{\frac{-3}{4\epsilon}}$. A straight-forward but tedious calculation (in following Lemma 13) shows that g is monotonically increasing for interval I and takes on values greater than $g(1/10)$ for any $\epsilon > 1/10$. Thus for $\epsilon > \frac{4}{3w}$ we get that $g(\epsilon) \geq g(\frac{4}{3w}) \geq q2^{-w}$.

We prove the remaining technical lemma to complete the proof of Corollary 1.

Lemma 13. *The function $g(\epsilon) := \epsilon - 2^{\frac{-3}{4\epsilon}}$ is monotonically increasing on $(0, 1/10] \subseteq \mathbb{R}$ and its minimum in the inveral $[1/10, 1] \subseteq \mathbb{R}$ is at $1/10$.*

Proof. Let $x = \epsilon^{-1}$, function $f(x) := x^{-1} - 2^{\frac{-3x}{4}}$ and intervals $I := [9, \infty) \subseteq \mathbb{R}$ and $J := [1, 9]$. Then if we can show that f is decreasing on I and greater than $f(10)$ on J then we are done.

The derivative is $f'(x) := -x^{-2} + 2^{\frac{-3x}{4}} \frac{3}{4} \ln 2$. We show that f' is negative on I. This is equivalent to showing that $\frac{4}{3 \ln 2} \geq x^2 2^{\frac{-3x}{4}}$ on I. Because x^2 grows more slowly then $2^{\frac{-3x}{4}}$ on I the right side of the inequality is decreasing on I. So by verifying that the inequality holds already for $x = 9$ we can conclude that it holds for all of I as desired.

Let function $h(x_1, x_2) := (x_1)^{-1} - 2^{\frac{-3x_2}{4}}$ then for any $a > b$ it holds that $h(a, b) \leq f(c)$ for all $c \in [a, b]$. To show the claim for interval J we fix 20 intervals $[l_i, l_{i+1}] \subset \mathbb{R}$ such their union contains $J \subset \cup_i [l_i, l_{i+1}]$ and for all of them $h(l_{i+1}, l_i) \geq f(10)$. Let $l_0 = 0$ and for $i \in [20]$ let $l_i = \left\lfloor (2^{-3l_{i-1}/4} + (1/10))^{-1} * 100 \right\rfloor / 100$. In particular we have the following values:

$l_0 = 1.00$	$l_1 = 1.43$	$l_2 = 1.73$	$l_3 = 1.97$	$l_4 = 2.17$	$l_5 = 2.36$	$l_6 = 2.54$
$l_7 = 2.72$	$l_8 = 2.91$	$l_9 = 3.12$	$l_{10} = 3.36$	$l_{11} = 3.64$	$l_{12} = 3.98$	$l_{13} = 4.41$
$l_{14} = 4.97$	$l_{15} = 5.69$	$l_{16} = 6.58$	$l_{17} = 7.53$	$l_{18} = 8.33$	$l_{19} = 8.83$	$l_{20} = 9.07$

and so J is contained in the union of the intervals. Moreover for any $i \in \mathbb{N}_{<20}$ we can write:

$$h(l_{i+1}, l_i) = (l_{i+1})^{-1} - 2^{\frac{-3l_i}{4}} \geq 2^{\frac{-3l_i}{4}} + (1/10) - 2^{\frac{-3l_i}{4}} = 1/10 > f(10)$$

□

By combining Corollary 1 and Theorem 1 we obtain our construction of a candidate MHF in the pROM.

Corollary 2. *Let $G = G_n : n \in \mathbb{N}$ be the graph from Theorem 1, $\mathbb{H}$ be a family of oracles and $f_n = f_{\mathbb{H}}^{G_n}$ be the associated graph functions. Then $f = \{f_n : n \in \mathbb{N}\}$ is an MHF in the pROM.*

In particular the non-triviality constraint of Definition 5 is satisfied by the algorithm T which simply computes each label in topological order storing all previously computed labels in it's current state. At most n labels will be stored (as G_n has n nodes), will never make more then a single oracle query per iteration and will finish in time exactly n.

Proof of Lemma 12. The remainder of this section is devoted to proving Lemma 12. For any $t \in \mathbb{N}$ define $c_t := \epsilon(\mathsf{cc}(G)(w - \log q) - t\overline{\epsilon})$ which is monotonically decreasing in t. We claim that an execution running in time t must have cost greater than both t and c_t. If this is the case then the lowest bound for an execution making at most q queries is obtained for runtime t_0 such that $t_0 = c_{t_0}$. Solving for t_0 we get exactly the bound in the lemma.

Since, by assumption, at least one bit is stored per iteration, trivially, any execution running for t iterations has cost at least t. So it remains to prove that the cost must also be larger than c_t. For this we adapting ideas from [DKW11,DNW05] to the CC setting. First we fix a method for converting an arbitrary execution in the pROM into a pebbling of G. That will allow us to prove that (with high probability) if the execution correctly computed $f_{\mathcal{H}}(x)$ then the resulting pebbling is both legal and complete. Moreover we show that (again with high probability) the CC of the pebbling can be upper-bounded using the cost of the execution. In particular with high probability the cost of execution can be lower-bound in terms of the CC of the graph as described by the definition of c_t.

Ex-Post-Facto Pebbling. Let G be a DAG, $\mathcal{H}$ be a random oracle, $\ell \in L$ and let lab be the $(\mathcal{H}, \ell)$ labeling of G. For convenience let $\mathsf{pre\text{-}lab}(v)$ denote $(v, \mathsf{lab}(v_1), \ldots, \mathsf{lab}(v_z))$ where the v_i are the parents of v arranged in lexicographic order. For arbitrary algorithm T we derive the *ex-post-facto* pebbling of G from (batch of) oracle calls $(\mathbf{q}_1, \ldots, \mathbf{q}_t)$ made during the execution of $T^{\mathcal{H}}(\ell)$.

An oracle call $\mathcal{H}(q)$ is called *correct* if it begins with some $v \in V$ such that $q = \mathsf{pre\text{-}lab}(v)$. In this case we call the parents of v the *input-nodes* of q and v the *output-node* of q. An oracle call is also correct if it has the form (v, ℓ, out) where out is some special symbol, v is a sink node and $\ell = \mathsf{lab}(v)$. Next we define the steps $(P_0, \ldots, P_t)$ of the ex-post-facto pebbling. We set $P_0 = \emptyset$ and define all other sets by going through the calls in the order they were made by T and applying the following rules to each batch $\mathbf{q}_i$.

1. For each query q if it is correct for some $v \in V$ then pebble v; that is add v to the set P_i.[20]
2. A node $v \in V$ is called *necessary* if there exists a $\mathbf{q}_j$ with $j > i$ containing a correct oracle call with v as an input-node but there is no $\mathbf{q}_k$ with $i < k < j$ containing a correct oracle call for v. Remove all v from P_i which are not necessary.[21]

We prove the following pair of claims about the ex-post-facto pebbling. The first states that it is legal with high probability while the second states that if T computed $f(x)$ correctly then with high probability the ex-post-facto pebbling is a complete pebbling for G and has at most a certain pebbling complexity which depends on $\mathsf{cost}_{[}\mathbb{H}](T, x)$. Intuitively these two properties imply the lemma since any algorithm computing f gives rise to a legal and complete pebbling of G with a limited cost. Yet, by assumption, any such pebbling must have at least a certain CC. Thus the algorithm must compute the function with a cost of at least a certain size.

Fix an arbitrary algorithm T, its coins, input x and oracle $\mathcal{H} \leftarrow \mathbb{H}$ and let $P = (P_1, \ldots, P_t)$ be the ex-post-facto pebbling associated with $T^{\mathcal{H}}(x)$.

Claim. Pebbling P is legal with probability at least $1 - \frac{q}{2^w}$ over the choice of $\mathcal{H}$ and the coins of T.

Proof. We prove the claim by contradiction. In particular assume that with probability greater than $\frac{q}{2^w}$ (over the choice of $\mathcal{O}$ and the coins of T) a pebble is placed on a node v (with positive in-degree) although its parents don't all have pebbles on them in the previous step. In other words a correct call for v is made by T with some w as input-node although no previous correct call was made to w as output-node. By assumption this must happen with greater than $\frac{q}{2^w}$. In this case we reach a contradiction by building a predictor which can predict at least one output value of $\mathcal{H}$ with impossibly high probability as follows. The predictor depends on input x and can query $\mathcal{H}$ at points of its choosing before outputting its prediction. Let r be an upper-bound on the number of random bits used by $T(x)$. The predictor also has access to a sequence of r uniform random bits it can use to simulate the random coins of T.

Hint: The predictor receives as a hint the index $i \in [q]$ of the (first) oracle call to v causing the illegal pebble placement in P.[22]

Execution: It runs $T(x)$ forwarding all oracle calls to $\mathcal{H}$ until the i^{th} query. By assumption this query is correct so it contains the labels of the parents of v at least one of which (say label ℓ_w for node w) was not previously queried to $\mathcal{H}$. The predictor can easily recompute the value of $\mathsf{pre\text{-}lab}(w)$ with out querying $\mathcal{H}(\mathsf{pre\text{-}lab}(w)$ so it outputs ℓ_w as its guess for the bits of $\mathcal{H}$ at position $\mathsf{pre\text{-}lab}(w)$.

[20] Placing a pebble on v will imply that T records its label in memory. Optimistically the reduction assumes no other values need to be stored.

[21] Intuitively a necessary node is one whose label will be needed in a future query but which will not be recomputed until then. Thus the labels of unnecessary nodes need not be stored.

[22] That is the value of the hint depends both on the choice $\mathcal{H}$ and the random bits made available to the predictor.

By assumption about the probability that the ex-post-facto pebbling is legal this predictor will succeed with probability greater then $\frac{q}{2^w}$ but this contradicts Lemma 1. □

Claim. For $i \in [t]$ let $c_i = |\sigma_i|$ be the bit-length of the state σ_i output by T at the end of iteration i. Then for any $\lambda \geq 0$ it holds:

$$\Pr\left[\forall i \in [t] \ : \ |P_i| \leq \frac{c_i + \lambda}{w - \log(q)}\right] > 1 - 2^{-\lambda}$$

over the choice of $\mathcal{H}$ and coins of T.

Proof. Recall that, intuitively, if $v \in P_i$ if it is necessary in iteration i. That is if, during some future iteration, T uses $\mathsf{lab}(v)$ as part of a (correct) query to the oracle without actually getting $\mathsf{lab}(v)$ from $\mathcal{H}$ again in the meantime. Therefor somehow T must able to reproduce the labels of all necessary nodes of iteration i during the later iterations using no more then the information it stored in σ_i. We formalize this intuition by giving the following predictor which predicts the label of all nodes in P_i leading to a contradiction of Lemma 1.

We name the oracle calls which cause a node $v \in P_i$ to be deemed necessary *critical* calls. There are at most $r \leq |P_i|$ critical calls. As before, the predictor depends on input x and has access to $\mathcal{H}$ and a long enough sequence of random bits used to simulate the coins of T.

Hint: The predictor receives as a hint the indices $J = \{j_1, \ldots, j_r\} \in [q]^r$ of the critical calls made by T and the state σ_i output by T at the end of iteration i.

Execution: It runs T on input (x, σ_i) recording the labels of all input-nodes of the critical calls. To answer any oracle call Q with output-node v the predictor does the following.

- Determines if the call is correct. A call is correct if and only if it is a critical call or for each parent w_i of v a correct call for w_i has already been made and Q matches the results of those calls. In particular it is easy to see that in this case $Q = \mathsf{pre\text{-}lab}(v)$ and no new calls need be made by the predictor to check this.
- If the call is correct and $\mathsf{lab}(v)$ has already been recorded then output it. Otherwise query $\mathcal{H}$ to answer the call.

When T terminates the predictor checks the transcript to determine the set P_i. It is easy to verify that their labels were never queried to $\mathcal{H}$. Thus for all $v \in P_i$ the predictor computes $\mathsf{pre\text{-}lab}(v)$ (using $\mathcal{H}$ and the recorded labels) and outputs $\mathsf{lab}(v)$ as its prediction of $\mathcal{H}(\mathsf{pre\text{-}lab}(v))$.

Now assume, for the sake of contradiction, that $\exists \lambda \geq 0$ such that with probability at least $2^{-\lambda}$ for some $i \in [t]$ it holds that $|P_i| > \frac{c_i + \lambda}{w - \log(q)}$ and let $\mathcal{W}$ be the event that the predictor succeeds. Then by construction $\Pr[\mathcal{W}] \geq 2^{-\lambda}$. On the other hand from Lemma 1 it follows that $\Pr[\mathcal{W}] \leq \frac{q^r * 2^{c_i}}{2^{|P_i|w}} \leq 2^{|P_i|(\log(q) - w) + c_i}$. In particular $2^{-\lambda} \leq 2^{|P_i|(\log(q) - w) + c_i}$ which is a contradiction to the assumption. □

We can now complete the proof of Lemma 12. Fix an input x and algorithm T making at most q oracle queries and let P be random variable describing the ex-post-facto pebblings for $T(x)$. We assume that whenever T appends a label ℓ for a sink $v \in V$ to its output register the query (v, ℓ, out) is added to the list of oracle queries made by T. While this has no effect on the cost of running T, it ensures that any correctly labeled sink node will receive a pebble in the ex-post-facto pebbling.

Now define $\mathcal{W}$ to be the event that T computes $f(x)$ correctly and assume that $\Pr[\mathcal{W}] \geq \epsilon$. Define $\mathcal{C}$ to be the event that P is legal and $\mathsf{p\text{-}cost}(P) \leq \frac{t\bar{\epsilon} + \sum c_i}{w - \log q}$ where t is the number of steps in P. Then the above two claims imply that $\Pr[\mathcal{C}] > 1 - q * 2^{-w} - 2^{-\bar{\epsilon}} = 1 - \epsilon$. Therefor the probability that both $\mathcal{W}$ and $\mathcal{C}$ occur simultaneously (in which case P is a complete pebbling of G) is positive. Thus it must be

that $\mathsf{cc}(G) \leq \frac{t\bar{\epsilon}+\sum c_i}{w-\log q}$ and so for any execution where T computes $f(x)$ correctly in time t its cumulative memory usage is $\sum c_i \geq \mathsf{cc}(G)(w - \log q) - t\bar{\epsilon}$.

Assuming, pessimistically, that whenever T fails to compute $f(x)$ it does so without incurring any cost we can conclude that for any x we have that any execution running in time t must have cost at least $\epsilon(\mathsf{cc}(G)(w - \log q) - t\bar{\epsilon}) = c_t$ which concludes the proof.

7 Acknowledgments

The first author would like to thank Krzysztof Pietrzak, Daniel Wichs, Stephan Krenn and Ueli Maurer for their patience and many valuable discussions.

References

ABFG13. Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. Cryptology ePrint Archive, Report 2013/805, 2013. `http://eprint.iacr.org/`.

Ada02. Adam Back. Hashcash - A Denial of Service Counter-Measure, 2002.

BDLHA13. Vitalik Buterin, Anthony Di Lorio, Charles Hoskinson, and Mihai Alisie. Ethereum: A Distributed Cryptographic Leger, 2013.

Ben89. Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

BM06. Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against aes. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop*, volume 4249 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2006.

Cha73. Ashok K. Chandra. Efficient compilation of linear recursive programs. In *SWAT (FOCS)*, pages 16–25. IEEE Computer Society, 1973.

Cha11. Charles Lee. Litecoin, 2011.

Cha13. Siu Man Chan. Just a pebble game. In *IEEE Conference on Computational Complexity*, pages 133–143. IEEE, 2013.

Coo73. Stephen A. Cook. An observation on time-storage trade off. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 29–33, New York, NY, USA, 1973. ACM.

DFKP13. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. Cryptology ePrint Archive, Report 2013/796, 2013. `http://eprint.iacr.org/`.

DGN03. Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 426–444. Springer, 2003.

DKW11. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 125–143. Springer, 2011.

DN93. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

DNW05. Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2005.

DT85. Patrick W. Dymond and Martin Tompa. Speedups of deterministic machines by synchronous parallel machines. *J. Comput. Syst. Sci.*, 30(2):149–161, 1985.

EGS75. Paul Erdoes, Ronald L. Graham, and Endre Szemeredi. On sparse graphs with dense long paths. Technical report, Stanford, CA, USA, 1975.

FLW13. Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A memory-consuming password scrambler. Cryptology ePrint Archive, Report 2013/525, 2013. `http://eprint.iacr.org/`.

FM98. Matthew K. Franklin and Dahlia Malkhi. Auditable metering with lightweight security. *Journal of Computer Security*, 6(4):237–256, 1998.

Fou98. Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998.

Gos12. Jeremi Gosney. Password Cracking HPC. Presented at Passwordˆ12 in Oslo, Norway, 2012.

HP70. Carl E. Hewitt and Michael S. Paterson. Record of the project mac conference on concurrent systems and parallel computation. chapter Comparative Schematology, pages 119–127. ACM, New York, NY, USA, 1970.

HPV77. John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977.

JB99. Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA*. The Internet Society, 1999.

JJ99. Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security*, CMS '99, pages 258–272, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

JM11. Yves Igor Jerschow and Martin Mauve. Non-parallelizable and non-interactive client puzzles from modular square roots. In *ARES*, pages 135–142. IEEE, 2011.

Kv10. Ghassan O. Karame and Srdjan Čapkun. Low-cost client puzzles based on modular exponentiation. In *Proceedings of the 15th European Conference on Research in Computer Security*, ESORICS'10, pages 679–697, Berlin, Heidelberg, 2010. Springer-Verlag.

LT82. Thomas Lengauer and Robert E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *J. ACM*, 29(4):1087–1130, October 1982.

Mig13. Miguel Freitas. Twister: Peer-to-Peer Microblogging, 2013.

MMV11. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2011.

MMV13. Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 373–388, New York, NY, USA, 2013. ACM.

Nak09. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

Nam11. Namecoin: A Secure Distributed Key/Value Storage System, 2011.

Pal13. Jackson Palmer. Dogecoin, 2013.

Per09. C. Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan 2009*, 2009.

Pip77. Nicholas Pippenger. Superconcentrators. *SIAM J. Comput.*, 6(2):298–304, 1977.

Pip80. Nicholas Pippenger. Comparative schematology and pebbling with auxiliary pushdowns (preliminary version). In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 351–356, New York, NY, USA, 1980. ACM.

PTC76. Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 149–160, New York, NY, USA, 1976. ACM.

RM99. Ran Raz and Pierre McKenzie. Separation of the monotone nc hierarchy. *Combinatorica*, 19(3):403–435, 1999.

RSW96. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.

Ruz79. Walter L. Ruzzo. Tree-size bounded alternation(extended abstract). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 352–359, New York, NY, USA, 1979. ACM.

Sav97. John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.

Sch83. Georg Schnitger. On depth-reduction and grates. In *FOCS*, pages 323–328. IEEE Computer Society, 1983.

Set75. Ravi Sethi. Complete register allocation problems. *SIAM J. Comput.*, 4(3):226–248, 1975.

SS78. John E. Savage and Sowmitri Swamy. Space-time trade-offs on the fft algorithm. *IEEE Transactions on Information Theory*, 24(5):563–568, 1978.

SS79a. John E. Savage and Sowmitri Swamy. Space-time tradeoffs for oblivious interger multiplications. In Hermann A. Maurer, editor, *ICALP*, volume 71 of *Lecture Notes in Computer Science*, pages 498–504. Springer, 1979.

SS79b. Sowmitri Swamy and John E. Savage. Space-time tradeoffs for linear recursion. In Alfred V. Aho, Stephen N. Zilles, and Barry K. Rosen, editors, *POPL*, pages 135–142. ACM Press, 1979.

TBFN07. Suratose Tritilanunt, Colin A. Boyd, Ernest Foo, and Juan M. Gonzalez Nieto. Toward non-parallelizable client puzzles. *Cryptology and Network Security (LNCS)*, 4856:247–264, 2007.

Tho79. C. D. Thompson. Area-time complexity for vlsi. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 81–88, New York, NY, USA, 1979. ACM.

Tom78. Martin Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 196–204, New York, NY, USA, 1978. ACM.

Val77. Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In Jozef Gruska, editor, *MFCS*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977.

VT89. H. Venkateswaran and Martin Tompa. A new pebble game that characterizes parallel complexity classes. *SIAM J. Comput.*, 18(3):533–549, 1989.

A Umbrealla Graphs.

We give a brief warm up for a naïve attempt at building a graph with high CC.

Lemma 4 shows that if a graph is to have any hope of having CC $O(n^2)$ then it must have depth $O(n)$. An umbrella graph (defined in a moment) is a naïve attempt at achieving $O(n^2)$ by augmenting a chain of length n with an edge structure that forces a large and evenly distributed number of the nodes to simultaneously contain a pebble in order to pebble the output node.

Lemma 14. *For any $n \in \mathbb{N}$ and positive divisor s let $A = \{i \in [n] : i \mod s = 0\}$. The umbrella graph $U_{s,n} = ([n], \{(i, i+1) : i \in [n-1]\} \cup \{(j, n) : j \in A\})$ has $\mathsf{cc}(U_{s,n}) = \Theta(n \log(n/s))$.*

Proof. Fix any optimal complete pebbling $P = (P_1, \ldots, P_t)$ of $U_{s,n}$. (In particular $P_t = \{n\}$.) We show that $\mathsf{p\text{-}cost}(P) = \Omega(n \log(n/s))$. The proof relies on the simple observation that if no pebble lies on the immediate $a \in [n]$ predecessors of a node in a chain then no pebble can be placed on the node can within the next a intervals. For intervals $i \in \{t-s, \ldots, t-1\}$ we have $|P_i| \geq n/s$ as otherwise A can not be fully pebbled by interval $t-1$. More generally for $j \in [n/s]$ and $i \in \{t - js, \ldots t - 1 - (j-1)s\}$ we have $|P_i| \geq n/(js)$ for the same reason. So we can write:

$$\sum_{i \in [t]} |P_i| \geq \sum_{j \in [n/s]} s * \left(\frac{n}{js}\right) = n \sum_{j \in [n/s]} \frac{1}{j} \geq n \ln\left(\frac{n}{s} + 1\right).$$

We can pebble $U_{s,n}$ to essentially meet this lower-bound. For the first $n/2$ iterations we move a single pebble out to node $n/2$, hence forth leaving a pebble on this node. For the next $n/4$ steps we move a pebble from 1 to $n/4$ and from $n/2$ to $(3n)/4$ again leaving them their for all subsequent iterations. We continue this recursively for n/s loops at which point all nodes in A (and only those) contain a pebble and we can finally pebble node n. A similar calculation as the one above shows that this approach indeed matches the lower-bound. □

We observe that an alternative construction where each node in a path of length n has incoming edges from all of it's predecessors in the path trivially results in optimal CC of $O(n^2)$. However as stated before we find this construction uninteresting as most applications of the pebbling paradigm consider only DAGs with constant in-degree.