Static-Memory-Hard Functions, and Modeling the Cost of Space vs. Time

Thaddeus Dryja Quanquan C. Liu Sunoo Park

MIT

Abstract

A series of recent research starting with (Alwen and Serbinenko, STOC 2015) has deepened our understanding of the notion of *memory-hardness* in cryptography — a useful property of hash functions for deterring large-scale password-cracking attacks — and has shown memory-hardness to have intricate connections with the theory of graph pebbling. Definitions of memory-hardness are not yet unified in the somewhat nascent field of memory-hardness, however, and the guarantees proven to date are with respect to a range of proposed definitions. In this paper, we observe *two* significant and practical considerations that are not analyzed by existing models of memory-hardness, and propose new models to capture them, accompanied by constructions based on new hard-to-pebble graphs. Our contribution is two-fold, as follows.

First, existing measures of memory-hardness only account for *dynamic* memory usage (i.e., memory read/written at runtime), and do not consider *static* memory usage (e.g., memory on disk). Among other things, this means that memory requirements considered by prior models are inherently upper-bounded by a hash function's runtime; in contrast, counting static memory would potentially allow quantification of much larger memory requirements, decoupled from runtime. We propose a new definition of *static-memory-hard* function (SHF) which takes static memory into account: we model static memory usage by oracle access to a large preprocessed string, which may be considered part of the hash function description. Static memory requirements are *complementary* to dynamic memory requirements: neither can replace the other, and to deter large-scale password-cracking attacks, a hash function will benefit from being *both* dynamic-memory-hard and static-memory-hard. We give two SHF constructions based on pebbling. To prove static-memory-hardness, we define a new pebble game ("*black-magic pebble game*"), and new graph constructions with optimal complexity under our proposed measure. Moreover, we provide a prototype implementation of our first SHF construction (which is based on pebbling of a simple "cylinder" graph), providing an initial demonstration of practical feasibility for a limited range of parameter settings.

Secondly, existing memory-hardness models implicitly assume that the cost of space and time are more or less on par: they consider only *linear* ratios between the costs of time and space. We propose a new model to capture *nonlinear* time-space trade-offs: e.g., how is the adversary impacted when space is quadratically more expensive than time? We prove that nonlinear tradeoffs can in fact cause adversaries to employ different strategies from linear tradeoffs.

Finally, as an additional contribution of independent interest, we present an asymptotically tight graph construction that achieves the best possible space complexity up to $\log \log n$-factors for an existing memory-hardness measure called *cumulative complexity* in the sequential pebbling model.

Contents

1 Introduction **3**

1.1 Background on graph pebbling . . . 5

1.2 Discussion on memory-hardness measures and related work . . . 5

1.3 Our contributions in more detail . . . 7

1.3.1 Static-memory-hard functions (SHFs) . . . 7

1.3.2 Remarks about the static-memory model . . . 8

1.3.3 Black-magic pebble game . . . 9

1.3.4 Capturing relative cost of memory vs. time . . . 10

1.4 Organization of the rest of the paper . . . 11

2 Pebbling definitions **11**

2.1 Standard and magic pebbling definitions . . . 11

2.2 Cost of pebbling . . . 13

2.2.1 Space complexity in standard pebbling . . . 13

2.2.2 Time complexity in standard pebbling . . . 14

2.2.3 Space complexity in black-magic pebbling . . . 14

2.3 Incrementally hard graphs . . . 15

2.3.1 α-tradeoff cumulative complexity . . . 15

3 Parallel random oracle model (PROM) **16**

3.1 Overview of PROM computation . . . 16

3.2 Functions defined by DAGs . . . 17

3.3 Relating complexity of PROM algorithms and pebbling strategies . . . 18

3.4 Legality and space usage of ex-post-facto black-magic pebbling . . . 19

4 Static-memory-hard functions **22**

4.1 Dynamic SHFs . . . 22

5 SHF constructions **23**

5.1 $\mathcal{H}_1$ constructions . . . 23

5.1.1 A failed attempt at $\mathcal{H}_1$ . . . 25

5.1.2 Cylinder construction . . . 26

5.1.3 Layering *shortcut-free* graphs . . . 29

5.2 $\mathcal{H}_2$ construction . . . 33

5.3 Proofs of hardness of SHF Constructions . . . 33

6 Capturing nonlinear space-time tradeoffs with CC$^\alpha$ **35**

6.1 CC and CC$^\alpha$ consider cumulative cost of *different strategies* . . . 36

6.2 Upper bounds for CC$^\alpha$ . . . 38

6.3 Asymptotically tight sequential lower bound for $\alpha = 1$ . . . 39

7 Cylinder-based SHF implementation **44**

7.1 Remarks on implementation and musings on random oracles in practice . . . 45

A Details of SHF construction with short labels **51**

B Regular and normal pebbling strategies **51**

1 Introduction

Pebble games were originally formulated to model time-space tradeoffs by a game played on DAGs. Generally, a DAG can be thought to represent a computation graph where each node is associated with some computation and a pebble placed on a node represents saving the result of its computation in memory. Thus, the number of pebbles represents the amount of memory necessary to perform some set of computations. The natural complexity measures to optimize in this game is the minimum number of pebbles used, as well as the minimum amount of time it takes to finish pebbling all the nodes; these goals correspond with minimizing the amount of memory and time of computation.

Pebble games were first introduced to study programming languages and compiler construction [PH70] but have since then been used to study a much broader range of tasks such as register allocation [Set75], proof complexity [AdRNV17, Nor12], time-space tradeoffs in Turing machine computation [Coo73, HPV77], reversible computation [Ben89], circuit complexity [Pot17], and time-space tradeoffs in various algorithms such as FFT [Tom81], linear recursion [Cha73, SS79b], matrix multiplication [Tom81], and integer multiplication [SS79a] in the RAM as well as the external memory model [JWK81]. To see a more comprehensive survey of the results in pebbling up to the last couple of years, see [Pip82] up to the 1980s and [Nor15] up to 2015.

The relationship between pebbling and cryptography has been a subject of research interest for decades, which has enjoyed renewed activity in the last few years. A series of recent works [AB16, ABH17, ABP17a, ABP17b, AS15, AT17, ACP$^+$16, AAC$^+$17, BZ16, BZ17] has deepened our understanding of the notion of *memory-hardness* in cryptography, and has shown memory-hardness to have intricate connections with the theory of graph pebbling.

Memory-hard functions (MHFs) have garnered substantial recent interest as a security measure against adversaries trying to perform attacks at scale, particularly in the ubiquitous context of password hashing. Consider the following scenario: hashes of user passwords are stored in a database,[1] and when a user enters a password p to log in, her computer sends $H(p)$ to the database server, and the server compares the received hash to its stored hash for that user's account. For a normal user, it would be no problem if hash evaluation were to take, say, one second. An attacker trying to guess the password by brute-force search, on the other hand, would try orders of magnitude more passwords, so a one-second hash evaluation could be prohibitively expensive for the attacker.

The evolution of password hashing functions has been something of an arms race for decades, starting with the ability to increase the number of rounds in the DES-based unix `crypt` function to increase its computation time—a feature that was used for exactly the above purpose of deterring large-scale password-cracking. Attackers responded by building special-purpose circuits for more efficient evaluation of `crypt`, resulting in a gap between the evaluation cost for an attacker and the cost for an honest user.[2]

A promising approach to mitigating this asymmetry in cost between hash evaluation on general- and special-purpose hardware is to increase the use of *memory* in the password hashing function. Memory is implemented in standardized ways which have been highly optimized, and memory chips are widely regarded to be an interchangeable commodity. Commonly used forms of memory — whether on-die SRAM cache, DRAM, or hard disks — are already optimized for the purpose of data I/O operations; and while there is active research in improving memory access times and costs, progress is and has been relatively incremental. This state of affairs sets up a relatively "even playing field," as the normal user and the attacker are likely to be using memory chips of similar memory access speed. While an attacker may choose to buy more memory, the cost of doing so scales linearly with the amount purchased.

[1]In practice, the password should first be concatenated with a random user-specific string called a *salt*, and then hashed. The salt is stored in the database alongside the hash to deter *dictionary attacks*.

[2]E.g., [CB02] discusses FPGA-based attacks on DES.

The designs of several MHFs proposed to date (e.g., [Per09, AS15, AB16, ACP+16, ABP17a]) have proven memory-hardness guarantees by basing their hash function constructions on DAGs, and using space complexity bounds from graph pebbling. Definitions of memory-hardness are not yet unified in this somewhat nascent field, however — the first MHF candidate was proposed only in 2009 [Per09] — and the guarantees proven are with respect to a range of definitions. The "cumulative complexity"-based definitions of [AS15] have enjoyed notable popularity, but some of their shortcomings have been pointed out by subsequent work proposing alternative more expressive measures, in particular, [ABP17b, AT17].

Our contribution We observe *two* significant and practical considerations not analyzed by existing models of memory-hardness, and propose new models to capture them, accompanied by constructions based on new hard-to-pebble graphs. Our main contribution is two-fold, as described in (1) and (2) below. We also provide an additional contribution of separate interest, described in (3).

1. ***Static-memory-hardness.*** Existing measures of memory-hardness only account for *dynamic* memory usage (i.e., memory read/written at runtime), and do not consider *static* memory usage (e.g., memory on disk). Among other things, this means that memory requirements considered by prior models are inherently upper-bounded by a hash function's runtime; in contrast, counting static memory would potentially allow quantification of much larger memory requirements, decoupled from the honest evaluator's runtime.

   We propose a new definition of *static-memory-hard* function (SHF) (Definition 4.2), and present two SHF constructions based on pebbling. To prove static-memory-hardness, we define a new pebble game called the *black-magic pebble game* (Definition 2.2), and prove properties of the space complexity of this game for new graphs (Graph Constructions 5.4 and 5.15). Graph Construction 5.15 gives rise to an SHF with a better asymptotic guarantee (same space usage but sustained over more time), whereas Graph Construction 5.4 yields an SHF with the advantage of simplicity in practice. Informal theorems stating the constructions' static-memory-hardness guarantees are given in Section 1.3 and formal theorems are in Section 5. In Section 7, we discuss our prototype implementation based on Graph Construction 5.4.

   We emphasize that static memory requirements are *complementary* to dynamic memory requirements: neither can replace the other, and to deter large-scale password-cracking attacks, a hash function will benefit from being *both* dynamic-memory-hard and static-memory-hard.
2. ***Modeling nonlinear cost of space vs. time.*** Existing measures of memory-hardness implicitly assume a linear trade-off between the costs of space and time. This model precludes situations where the relative costs of space and time might be more unbalanced (e.g., quadratic or cubic). We demonstrate that this modeling limitation is significant, by giving an example where adversaries facing asymptotically different space-time cost tradeoffs would in fact employ *different strategies.* Then, to remedy this shortcoming, we define *graph-optimal* variants of memory-hardness measures (in Section 2) that *explicitly* model the relative cost of space and time. These can be seen as extending the main memory-hardness measures in the literature (namely, *cumulative complexity* and *sustained memory complexity*). We prove bounds on the new measure as elaborated in Section 1.3.
3. We give the first graph construction that is tight, up to $\log\log n$-factors, to the optimal cumulative complexity that can be achieved for any graph (upper bound due to [ABP17a, ABP17b]).

   Informal version of Theorem 6.23. There exists a family of graphs where the cumulative complexity of any constant in-degree graph with n nodes in the family is $\Theta\left(\frac{n^2 \log\log n}{\log n}\right)$ which is asymptotically tight to the upper bound of $\Theta\left(\frac{n^2 \log\log n}{\log n}\right)$ given in [ABP17a, ABP17b] in the sequential pebbling model.

Next, Section 1.1 gives a brief background on graph pebbling, Section 1.2 gives discussion on memory-hardness measures and related work, and Sections 1.3.1 and 1.3.4 give more detailed high-level overviews of our SHF contribution and nonlinear space-time tradeoff model (items (1) and (2) above), respectively.

1.1 Background on graph pebbling

The standard *black pebble game* is parametrized by a directed acyclic graph (DAG) and a special subset of its nodes (called the *target set*). In the game, an unlimited supply of "pebbles" is made available to a player, who must place and remove pebbles on the nodes of the DAG in a sequence of moves according to the following two rules.

1. A pebble may be placed or moved onto a node only if all of its predecessors have already been pebbled. (In particular, pebbles may be placed on source nodes at any time.)
2. Any pebble can be removed from the graph at any time.

The goal of the game is to arrive at a state where every pebble in the target set is covered by a pebble. Often, the target set is the set of the sink nodes.

The pebbling literature, starting with [PH70, Set75, Coo73, HPV77], has established a number of complexity measures describing the complexity of pebbling: e.g., measuring the minimum number of pebbles that must be used to achieve a complete pebbling, or the minimum number of moves needed. In the literature, there are several variants of the game, including sequential and parallel (depending on whether many pebbles can be placed in a single move), and versions where other different types of pebbles are used (such as the red-blue pebble game [JWK81] and the black-white pebble game [CS74]). In this work, our results are stated and proven in the context of constant in-degree graphs for simplicity; however, most of our results extend straightforwardly to non-constant in-degree graphs.

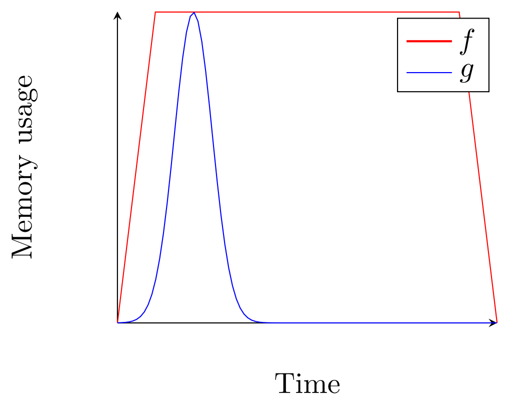
Graph pebbling and memory-hardness Graph pebbling algorithms can be used to construct hash functions in the (parallel) random oracle model. This paradigm has been used by prior constructions of memory-hard hashing [AS15] as well as other prior works [DKW11].

Informally, the idea to "convert" a graph into a hash function is to associate with each node v a string called a *label*, which is defined to be $\mathcal{O}(v, \mathsf{pred}(v))$ where $\mathcal{O}$ is a random oracle and $\mathsf{pred}(v)$ is the list of labels of predecessors of v. For source nodes, the label is instead defined to be $\mathcal{O}(v, \zeta)$ for a string ζ which is an input to the hash function. The output of the hash function is defined to be the list of labels of target nodes. Intuitively, since the label of a node cannot be computed without the "random" labels of all its predecessors, any algorithm computing this hash function must move through the nodes of the graph according to rules very similar to those prescribed by the pebbling game; and therefore, the memory requirement of computing the hash function roughly corresponds to the pebble requirement of the graph. Thus, proving lower bounds on the pebbling complexity of graph families has useful implications for constructing provably memory-hard functions.

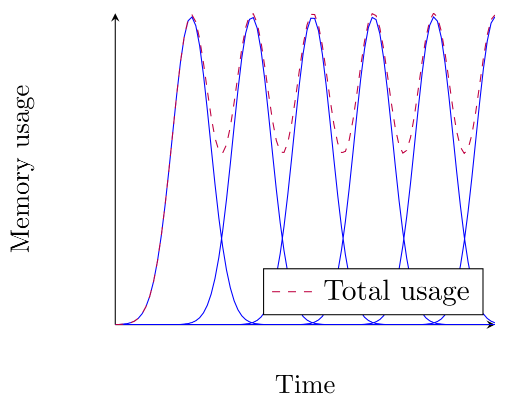
In our setting, in contrast to previous work, we employ a variant of the above technique: the string ζ is a fixed parameter of our hash function, and the input to the hash function instead specifies the indices of the target nodes whose labels are to be outputted.

1.2 Discussion on memory-hardness measures and related work

The original paper proposing memory-hard functions [Per09] suggested a very simple measure: the minimum amount of memory necessary to compute the hash function. It was subsequently observed that a major drawback of this measure is that it does not distinguish between functions f and g with the same peak memory usage, even if the peak memory lasts a long time in evaluating f and is just fleeting in evaluating g (Figure 1a). This is significant as the latter type of function is much better for a password-cracking adversary. In particular, pipelining the evaluation of the latter type of function would allow reuse of the same memory for many function evaluations at once, effectively reducing the adversary's amortized memory requirement by a factor of the number of concurrent executions (Figure 1b).



(a) Functions with the same peak memory usage



(b) Pipelined evaluations of g (reusing memory)

Figure 1: Limitations of peak memory usage as a memory-hardness measure

Cumulative complexity [AS15] put forward the notion of *cumulative complexity* (CC), a complexity measure on graphs. CC was adopted by several subsequent works as a canonical measure of memory-hardness. CC measures the *cumulative* memory usage of a graph pebbling function evaluation: that is, the sum of memory usage over all time-steps of computation. In other words, this is the area under a graph of memory usage against time. CC is designed to be very robust against amortization, and in particular, scales linearly when computing many copies of a function on different inputs. This is a great advantage compared to the simpler measure of [Per09], which does not account well for an amortizing adversary (as shown in Figure 1).

Depth-robust graphs More recently, [AB16, ABP17a] proved bounds on optimal CC of certain graph families. They showed that a particular graph property called *depth-robustness* suffices to attain optimal CC (up to polylog factors–the CC of any graph with bounded in-degree is upper bounded by $O\left(\frac{n^2 \log\log n}{\log n}\right)$ [AB16, ABP17b]). An (r, s)-depth-robust graph is one where there exists a path of length s even when any r vertices are removed. Intuitively, this captures the notion that storing any r vertices of the graph will not shortcut the pebbling in a significant way. It turns out that depth-robustness will again be a useful property in our new model of memory-hardness with preprocessing.

Sustained memory complexity Very recently, Alwen, Blocki, and Pietrzak [ABP17a] proposed a new measure of memory complexity, which captures not only the cumulative memory usage over time (as does CC), but goes further and captures the amount of time for which a particular level of memory usage is sustained. Our SHF definition also captures *sustained* memory usage: we propose a definition of capturing the duration for which a given amount of memory is required, designed to capture static as well as dynamic memory requirements. By the nature of static memory, it is especially appropriate in our setting to consider (and maximize) the amount of time for which a static memory requirement is *sustained*.

Core-area memory ratio Previous works have considered certain hardware-dependent non-linearities in the ratio between the cost of memory and computation [BK15, AB16, RD17]. Such phenomena may incur a multiplicative factor increase in the memory cost that is dependent, in a possibly non-linear way, on specific hardware features. Note that *the non-linearity here is in the hardware-dependence*, rather than the space-time tradeoff itself. In contrast, our new models are more expressive, in that they make configurable the asymptotic tradeoff between space and time (by a parameter α which is in the exponent, as detailed in Definition 2.16) in an application-dependent way. This versatility of configuration targets applications where the trade-off may realistically depend on arbitrary and possibly exogenous space/time costs, and thus contrasts with metrics tailored for a specific hardware feature, such as core-memory ratio.

Towards a general theory of moderately hard functions Most recently, Alwen and Tackmann [AT17] proposed a more general (though not comprehensive) framework for defining desirable guarantees of "moderately hard functions," i.e., functions that are efficient to compute but somewhat hard to invert. Their work points out a number of drawbacks of prior measures such as those described above. Notably, many of the prior measures characterized the hardness of *computing* the function with an implicit assumption that this hardness would translate to the hardness of *inverting* the function (as it would indeed in the case of a brute-force approach to inversion). In other words, these measures implicitly assume that the hash function in question "behaves like a random oracle" in the sense that brute-force inversion is the optimal approach.

1.3 Our contributions in more detail

To prove static-memory-hardness, we define a new pebble game called the *black-magic pebble game* (Definition 2.2), and prove properties of the space complexity of this game for new graphs (Graph Constructions 5.4 and 5.15).

The black-magic pebble game may additionally be of independent interest for the pebbling literature. Indeed, a pebble game used to analyze security of *proofs of space* [DFKP15] can be viewed as a non-adaptive[3] version of the black-magic pebble game in which the target node set is sampled from a distribution by a challenger.

Based on our new graph constructions, we construct SHFs with provable guarantees on sustained memory usage, as follows. Graph Construction 5.15 gives a better asymptotic guarantee (same space usage but sustained over more time), whereas Graph Construction 5.4 has the advantage of simplicity in practice. In Section 7, we discuss our prototype implementation based on Graph Construction 5.4.

1.3.1 Static-memory-hard functions (SHFs)

Prior memory-hardness measures make a modeling assumption: namely, that the memory usage of interest is solely that of memory dynamically generated at run-time. However, static memory can be costly for the adversary too, and yet it is not taken into account by existing measures such as CC. Intuitively, it can be beneficial to design a function whose evaluation requires keeping a large amount of static memory on disk (which may be thought to be produced in a one-time initial setup phase). While not all the static memory might be accessed in any given evaluation, the "necessity" to maintain the data on disk can arise from the idea that an adversary attempting to evaluate the function on an arbitrary input while having stored a lesser amount of data would be forced to *dynamically* generate comparable amounts of memory. Note that the resulting *dynamic* memory requirements could be orders of magnitude larger (say, gigabytes) than the memory requirements of

[3]Here, "non-adaptive" means that all magic pebbles must be fixed at the start of the game rather than placed throughout the game.

existing memory-hard function proposals, because unlike in prior memory-hardness models, here we have decoupled the memory requirement from the memory requirements of the honest evaluator.

We propose a new model and definitions for *static-memory-hard functions* (SHFs), in which we model static memory usage by oracle access to a large preprocessed string, which may be considered part of the hash function description. In particular, the preprocessed string can be public and known to the adversary — the important guarantee is that without storing (almost) all of it statically, the adversary will incur huge online memory requirements.

Definition (informal). We model a *static-memory-hard function family* as a two-part algorithm $\mathcal{H} = (\mathcal{H}_1, \mathcal{H}_2)$ in the parallel random oracle model, where $\mathcal{H}_1(1^\kappa)$ outputs a "large" string to which $\mathcal{H}_2$ has oracle access,[4] and $\mathcal{H}_2$ receives an input x and outputs a hash function output y. Informally, our hardness requirement is that with high probability, any *two-part* adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ must *either* have $\mathcal{A}_1$ output a large state (comparable to the output size of $\mathcal{H}_1$), *or* have $\mathcal{A}_2$ use large (dynamic) space.

We then give two constructions of SHFs based on graph pebbling. To prove static-memory-hardness, we define a new pebble game called the *black-magic pebble game* of which we give an overview in Section 1.3.3. Our simpler SHF construction is based on a family of tree-like "cylinder" graphs, which achieves memory usage proportional to the square root of the number of nodes, sustained over time proportional to the square root of the number of nodes. Furthermore, we give a better construction based on pebbling of a new graph family, that achieves better parameters: the same (square root) memory usage, but sustained over time proportional to the number of nodes.

Informal version of Theorem 5.28. The "cylinder graph" (Graph Construction 5.4) can be used to construct an SHF with static memory requirement $\Lambda \in \Theta(\sqrt{n}/(\kappa - \xi\log(\kappa))$ where n is the number of nodes in the graph, κ is a security parameter, and $\xi \in \omega(1)$, such that any adversary using non-trivially less *static* memory than Λ must incur at least Λ *dynamic* memory usage for at least $\Theta(\sqrt{n})$ steps.

Informal version of Theorem 5.29. Graph Construction 5.15 can be used to construct an SHF with static memory requirement $\Lambda \in \Theta(\sqrt{n})/(\kappa - \xi\log(\kappa))$ where n, κ, and ξ are as described above, such that any adversary using non-trivially less *static* memory than Λ must incur at least Λ *dynamic* memory usage for at least $\Theta(n)$ steps.

Static memory requirements are *complementary* to dynamic memory requirements: neither can replace the other, and to deter large-scale password-cracking attacks, a hash function will benefit from being *both* dynamic-memory-hard and static-memory-hard. In Section 4.1, we give a discussion of how, given a static-memory-hard function and a (dynamic-)memory-hard function, they can be concatenated to yield a "dynamic SHF" that inherits both the static memory requirement of the former and the dynamic memory requirement of the latter.

Implementation We have a prototype implementation of our "cylinder" SHF construction. The code is available on github at `https://github.com/adiabat/masshash`. A discussion of the implementation and its performance for different static memory sizes is given in Section 7.

1.3.2 Remarks about the static-memory model

On static vs. dynamic memory In the case of function performing lookups to a large, static memory table, the memory storing the table does not need to be writable. This may seem to point to an optimization for the attacker: produce a read-only memory chip which supports fast, random-access read queries, but omits the hardware needed for writing data, as it has been pre-programmed

[4]More precisely, $\mathcal{H}_2$ may adaptively query the value of $\mathcal{H}_1$'s output string at specific locations.

at the factory with the precomputed static table. However, in practice, this optimization seems implausible. In modern hardware, ROM chips have almost entirely disappeared; where they do still exist, they are used for their non-volatile storage properties (they retain data when power is lost, unlike most RAM), and are copied to RAM before being read from, due to the low speed of the ROM. Current development focuses almost exclusively on dynamic access memory which supports both reads and writes, so it is reasonable to believe that an attacker would need to use this type of hardware; switching to ROM would likely increase costs and slow down access to the static table.

Static and dynamic memory requirements are thus incomparable, and both are useful to deter a password-cracking adversary.

Alternative application: bounded retrieval ("big-key") model As already stated above, the preprocessed string in our setting is assumed to be public, and our static-memory-hardness guarantees hold assuming the adversary knows the string. This is useful as it allows defining a single hash function accessible to all parties in a system, like a random oracle: one could imagine a standards body like NIST simply publishing a set of parameters defining a fixed hash function with a fixed "preprocessed string." One informal way to think of this is that the preprocessed string is part of the description of a fixed hash function. A single hash function accessible to everyone is particularly useful for certain applications such as checksums, where many parties in a distributed network may need to compute the same hash function.

In some other applications, however, hash function *families* may suffice or be more appropriate, i.e., where each party samples a function from the family for her own use, rather than every party using exactly the same function. In such applications, the preprocessed string can be considered the *seed* of a particular hash function from the family defined by $(\mathcal{H}_1, \mathcal{H}_2)$, and generated on a per-application basis. We observe one potential advantage of such a setup, inspired by the bounded retrieval [Dzi06, CLW06, CDD+07, ADW09, ADN+10, ADW09] ("big-key" [BKR16]) model.[5]: to make hash function evaluation more difficult for (e.g., password-cracking) adversaries. If the party using the hash function decides to keep the preprocessed string *secret*, then an adversary would have to exfiltrate almost all of the large preprocessed string from the honest user in order to be able to evaluate the hash function. As observed in the bounded retrieval literature, exfiltrating large quantities of data (say, gigabytes) can be much more costly for adversaries than exfiltrating smaller data items (such as secret keys).

1.3.3 Black-magic pebble game

We introduce a new pebble game called the *black-magic pebble game*. This game bears some similarity to the standard (black) pebble game, with the main difference that the player has access to an additional set of pebbles called *magic pebbles*. Magic pebbles are subject to different rules from standard pebbles: they may be placed anywhere at any time, but cannot be removed once placed, and may be limited in supply. The pebbling space cost of this game is defined as the maximum number of standard pebbles on the graph at any time-step plus the total number of magic pebbles used throughout the computation. Observe that while the most time-efficient strategy in the black-magic pebble game is always to pebble all the target nodes with magic pebbles in the first step, the most space-efficient strategy is much less clear.

Lower-bounds on space usage can be non-trivially different between the standard and magic pebbling games. For example, if a graph has a constant number of targets, then magic pebbling space usage will never exceed a constant number of pebbles, whereas the standard pebbling space usage can be super-constant. In particular, it is unclear, in the new setting of magic pebbling, whether

[5] We cite the seminal papers that coined these terms, and note that there has been a rich literature on the topic since.

known lower-bounds on pebbling space usage in the standard pebble game[6] are transferable to the magic pebble game. We prove in Section 5 that for layered graphs,[7] the best possible lower-bound for the magic pebbling game is $\Theta(\sqrt{n})$.

We leave determining the lower bound for magic pebbling space usage in general graphs as an open question. An answer to this open question would be useful towards constructing better static-memory-hard functions using the paradigm presented herein.

Our proof techniques rely on a close relationship between black-magic pebbling complexity and a new graph property which we define, called *local hardness*. Local hardness considers black-magic pebbling complexity in a variant model where *subsets* of target nodes are required to be pebbled (rather than *all* target nodes, as in the traditional pebbling game), and moreover, a "preprocessing phase" is allowed, wherein magic pebbles may be placed on the graph in advance of knowing which target nodes are to be produced. This "preprocessing" aspect bears some resemblance to the *black-white pebbling game* [CS74], a variant of the standard pebbling game in which some limited number of *white* pebbles can be placed "for free," and the black pebbles must be placed according the standard rules. However, our setting differs from the black-white pebbling game: while preprocessing and storing magic pebbles in advance can be viewed as analogous to placing white pebbles for free, the black-white pebbling game imposes restrictions on the *removal* of white pebbles from the graph, which are not present in our setting.

1.3.4 Capturing relative cost of memory vs. time

Existing measures such as CC and sustained memory complexity trade off space against time at a linear ratio. In particular, CC measures the minimal area under a graph of memory usage against time, over all possible algorithms that evaluate a function.[8]

However, different applications may have different relative cost of space and time. We propose and define a variant of CC called α-CC, parametrized by α which determines the relative cost of space and time, and observe that α-CC may be meaningfully different from CC and more suitable for certain application scenarios. For example, when memory is "quadratically" more expensive than time, the measure of interest to an adversary may be the area under a graph of memory squared against time, as demonstrated by the following theorem.

Informal version of Theorem 6.8. There exist graphs for which an adversary facing a linear space-time cost trade-off would in fact employ a *different pebbling strategy* from one facing a cubic trade-off.

It follows that when the costs of space and time are not linearly related, the CC measure may be measuring the complexity of *the wrong algorithm*, i.e., not the algorithm that an adversary would in fact favor. We thus see that our α-CC measure is more appropriate in settings where space may be substantially more costly than time (or vice versa). Moreover, our parametrized approach generalizes naturally to sustained memory complexity. We show that our graph constructions are invariant across different values of α, a potentially desirable property for hash functions so that they are robust against different types of adversaries.

[6]E.g., $\Theta\left(\frac{n}{\log n}\right)$ space is necessary to pebble certain classes of graphs in the standard pebble game [LT82].

[7]"Layered graph" is a standard term in the pebbling literature that refers to graphs whose nodes can be partitioned into a sequence of "layers" such that edges only go between vertices in adjacent layers.

[8]Of course, in general, memory usage and time depend on the specific computational model in discussion. However, in the stylized parallel random oracle model (PROM), on which all analyses in this paper (and previous literature on MHFs) are based, time-steps and memory usage are well-defined. We refer to Section 3 for a description of the PROM.

Informal version of Theorem 6.13. Given any graph construction $G = (V, E)$, there exists a pebbling strategy that is less expensive asymptotically than any strategy using a number of pebbles asymptotically equal to the number of nodes in the graph for any time-space tradeoff.

1.4 Organization of the rest of the paper

Section 2 introduces *standard and new* graph pebbling definitions, Section 3 introduces computation in the parallel random oracle model and its relation to our new *black-magic* pebbling complexity measures, Section 4 introduces our definition of a static-memory-hard function (SHF), Section 5 gives our SHF constructions and proofs. Then, Section 6 presents our modeling and motivation of nonlinear cost tradeoffs between space and time, with upper and lower bounds in the new model. Finally, Section 7 discusses our prototype SHF implementation.

2 Pebbling definitions

A *pebbling game* is a one-player game played on a DAG where the goal of the player is to place pebbles on a set of one or more *target nodes* in the DAG.

In Section 2.1, we formally define two variations of the sequential and parallel pebble games: the *standard (black) pebble game* and the *black-magic pebble game*, the latter of which we introduce in this work. We also give the definitions of valid strategies and moves in these games. Then in Section 2.2, we define measures for evaluating the sequential and parallel pebbling complexity on families of graphs.

2.1 Standard and magic pebbling definitions

Definition 2.1 (Standard (black) pebble game)**.**

- ***Input:*** *A DAG, $G = (V, E)$, and a* target set $T \subseteq V$. *Define* $\mathsf{pred}(v) = \{u \in V : (u, v) \in E\}$, *and let $S \subseteq V$ be the set of sources of G.*
- ***Rules at move i:*** *At the start of the game, no node of G contains a pebble. The player has access to a supply of* black pebbles. *Game-play proceeds in discrete moves, and P_i (called a "pebble configuration") is defined as the set of nodes containing pebbles after the ith move. $P_0 = \varnothing$ represents the initial configuration where no pebbles have been placed. Each move may consist of multiple actions adhering to the following rules.*[9]
  1. *A pebble can be placed on any source, $s \in S$.*
  2. *A pebble can be removed from any vertex.*
  3. *A pebble can be placed on a non-source vertex, v, if and only if its direct predecessors were pebbled at time $i - 1$ (i.e.,* $\mathsf{pred}(v) \in P_{i-1}$*).*
  4. *A pebble can be moved from vertex v to vertex w if and only if $(v, w) \in E$ and* $\mathsf{pred}(w) \in P_{i-1}$.
- ***Goal:*** *Pebble all nodes in T at least once (i.e., $T \subseteq \bigcup_{i=0}^{t} P_i$).*[10]

Remark. At first glance, it may seem that rule 4 in Definition 2.1 is redundant as a similar effect can be achieved by a combination of the other rules. However, the application of rule 4 can allow the usage of fewer pebbles. For example, a simple two-layer binary tree (with three nodes) could

[9]Multiple applications of rules 1, 2, and 3 can occur in a single move. E.g., multiple sources can be pebbled in a single move. Rule 4 can also be applied multiple times in a single move *for different pebbles*, but cannot be applied more than once to the same pebble (since, naturally, a single pebble cannot move to multiple locations).

[10]This goal statement corresponds to the notion of a *visiting pebbling* as defined in [Nor15]. Our paper will use this *visiting pebbling* notion throughout; however, we remark that an alternative notion of pebbling exists in the literature, called *persistent pebbling*, which requires that all the nodes in T be pebbled in the final configuration (i.e., $T \subseteq P_t$).

be pebbled with two pebbles using rule 4, but would require three pebbles otherwise. Nordstrom [Nor15] showed that in sequential strategies, it is always possible to use one fewer pebble by using rule 4.

We note for completeness that while rule 4 is standard in the pebbling literature, not all the papers in the MHF literature include rule 4.

Next, we define the *black-magic pebble game* which we will use to prove security properties of our static-memory-hard functions.

Definition 2.2 (Black-magic pebble game).

- ***Input:*** *A DAG $G = (V, E)$, a target set $T \subseteq V$, and* magic pebble bound $\mathfrak{M} \in \mathbb{N} \cup \{\infty\}$.
- ***Rules:*** *At the start of the game, no node of G contains a pebble. The player has access to two types of pebbles:* black pebbles *and up to $\mathfrak{M}$* magic pebbles. *Game-play proceeds in discrete moves, and $P_i = (M_i, B_i)$ is the pebble configuration after the ith move, where M_i, B_i are the sets of nodes containing magic and black pebbles after the ith move, respectively. $P_0 = (\varnothing, \varnothing)$ represents the initial configuration where no black pebbles or magic pebbles have been placed. Each move may consist of multiple actions adhering to the following rules.*
  1. *Black pebbles can be placed and removed according to the rules of the standard pebble game which are defined in the full version.*[11]
  2. *A magic pebble can be placed on and removed from any node, subject to the constraint that at most $\mathfrak{M}$ magic pebbles are used throughout the game.*
  3. *Each magic pebble can be placed at most once: after a magic pebble is removed from a node, it disappears and can never be used again.*
- ***Goal:*** *Pebble all nodes in T at least once (i.e., $T \subseteq \bigcup_{i=0}^{t} (M_i \cup B_i)$).*

Remark. In the black-magic pebble game, unlike in the standard pebble game, there is always the simple strategy of placing magic pebbles directly on all the target nodes. At first glance, this may seem to trivialize the black-magic game. When optimizing for space usage, however, this simple strategy may not be favorable for the player: by employing a different strategy, the player might be able to use much fewer than T pebbles overall.

Next, we define valid sequential and parallel strategies in these games.

Definition 2.3 (Pebbling strategy). *Let G be a graph and T be a target set. A* standard (resp., black-magic) pebbling strategy for (G, T) *is defined as a sequence of pebble configurations, $\mathcal{P} = \{P_0, \dots, P_t\}$, satisfying conditions 1 and 2 below. $\mathcal{P}$ is moreover* valid *if it satisfies condition 3, and* sequential *if it satisfies condition 4.*

1. *$P_0 = \varnothing$.*
2. *For each $i \in [t]$, P_i can be obtained from P_{i-1} by a legal move in the standard (resp., black-magic) pebble game.*
3. *$\mathcal{P}$ successfully pebbles all targets, i.e., $T \subseteq \bigcup\limits_{i=0}^{t} P_i$.*
4. *For each $i \in [t]$, P_i contains at most one vertex not contained in P_{i-1} (i.e., $|P_i \setminus P_{i-1}| \leq 1$).*

A black-magic pebbling strategy must satisfy one additional condition to be considered valid*:*

5. *At most $\mathfrak{M}$ magic pebbles are used throughout the strategy, i.e., $|\bigcup_{i \in [t]} M_i| \leq \mathfrak{M}$ where M_i is the ith configuration of magic pebbles.*

[11]The rules of the standard pebble game are a standard definition in the pebbling literature. In the black-magic game, a predecessor node counts as "pebbled" if it contains either a black or a magic pebble. Where Definition 2.1 treats P_i as a set of nodes, Definition 2.2 treats P_i as equal to $M_i \cup B_i$.

2.2 Cost of pebbling

In this subsection, we give definitions of several cost measures of graph pebbling, applicable to the standard and black-magic pebbling games. While these definitions assume parallel strategies, we note that the sequential versions of the definitions are entirely analogous.

2.2.1 Space complexity in standard pebbling

We give a brief informal summary of the definitions in this subsection, before proceeding to the formal definitions.

Pebbling complexity measures We informally overview the pebbling complexity definitions, some of which are new to this work.

The *time complexity* of a pebbling strategy $\mathcal{P}$ is the number of steps, i.e., $\mathsf{Time}\,(\mathcal{P}) = |\mathcal{P}|$. The *time complexity* of a graph $G = (V, E)$ given that at most S pebbles can be used is $\mathsf{Time}(G, S) = \min_{\mathcal{P}\in\mathbb{P}_{G,T,S}} (\mathsf{Time}\,(\mathcal{P}))$. Next, we overview variants of space complexity.

1. **Space complexity** of a *pebbling strategy* $\mathcal{P}$ on a graph G, denoted by $\mathbf{P}_{\mathrm{s}}(\mathcal{P})$, is the minimum number of pebbles required to execute $\mathcal{P}$. Space complexity of the *graph* G with target set T, written $\mathbf{P}_{\mathrm{s}}(G, T)$, is the minimum space complexity of any valid pebbling strategy for G.
2. **Λ-sustained space complexity** [ABP17a][12] of a *pebbling strategy* $\mathcal{P}$ on a graph G, denoted by $\mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \Lambda)$, is the number of time-steps during the execution of $\mathcal{P}$, in which at least Λ pebbles are used. Λ-sustained space complexity of the *graph* G with target set T, written $\mathbf{P}_{\mathrm{ss}}(G, \Lambda, T)$ is the minimum Λ-sustained space complexity of all valid pebbling strategies for G.
3. **Graph-optimal sustained complexity** of a *pebbling strategy* $\mathcal{P}$, denoted by $\mathbf{P}_{\text{opt-ss}}(\mathcal{P})$, is the number of time-steps during the execution of $\mathcal{P}$, in which the number of pebbles in use is equal to the space complexity of G. Graph-optimal sustained complexity of the *graph* G with target set T, written $\mathbf{P}_{\text{opt-ss}}(G, T)$ is the minimum graph-optimal sustained complexity of all valid pebbling strategies for G.
4. **Δ-suboptimal sustained complexity** of a *pebbling strategy* $\mathcal{P}$ is the number of time-steps, during the execution of $\mathcal{P}$, in which the number of pebbles in use is at least the space complexity of G *minus* Δ. Δ-suboptimal sustained complexity of the *graph* G is the minimum Δ-suboptimal sustained complexity of all valid pebbling strategies for G.

A couple of remarks are in order.

Remark. The third and fourth definitions are new to this paper. They can be seen as special variants of Λ-sustained space complexity, i.e., with a special setting of Λ dependent on the specific graph family in question. They are useful to define in their own right, as unlike plain Λ-sustained space complexity, these measures express complexity for a given graph family relative to the best possible value of Λ at which sustained space usage could be hoped for. In the rest of this paper, we prove guarantees on *graph-optimal sustained complexity* of our constructions, which have high sustained space usage at the optimal Λ-value. However, we also define Δ*-suboptimal sustained complexity* here for completeness, since it is more general[13] and preferable to graph-optimal complexity when evaluating graph families where the maximal space usage may not be sustained for very long.

[12]We note that our notation diverges from that of [ABP17a], but our Definition 2.5 is equivalent to their definition of "s-sustained space complexity." (E.g., they write $\Pi_{ss}(\mathcal{P}, \Lambda)$ instead of $\mathbf{P}_{\mathrm{ss}}(G, \mathcal{P}, \Lambda)$.) We gave this decision some consideration as inconsistent notation can add confusing clutter to a literature; we decided on our notation (1) in order to keep consistency with the pebbling literature, where the pyramid graphs that will be used in our SHF construction are traditionally denoted by Π; and (2) because our notation makes the graph G explicit where sometimes it is implicit in [ABP17a], and this is important for the new "graph-optimal sustained complexity" notion we introduce.

[13]More specifically, graph-optimal sustained complexity is Δ-suboptimal sustained complexity for $\Delta = 0$.

Remark. We have found the term "Λ-sustained space complexity" can be slightly confusing, in that it measures a number of time-steps rather than an amount of space. We retain the original terminology as it was introduced, but include this remark to clarify this point.

We now present the formal definitions of the complexity measures for the standard pebbling game. In all of the below definitions, $G = (V, E)$ is a graph, $T \subseteq V$ is a target set, $\mathcal{P} = (P_1, \ldots, P_t)$ is a standard pebbling strategy on (G, T), and $\mathbb{P}_{G,T}$ denotes the set of all valid standard pebbling strategies on (G, T).

Definition 2.4. *The* space complexity of pebbling strategy $\mathcal{P}$ *is:* $\mathbf{P}_{\mathrm{s}}(\mathcal{P}) = \max_{P_i \in \mathcal{P}} (|P_i|)$. *The* space complexity of G *is the minimal space complexity of any valid pebbling strategy that pebbles the target set* $T \subset V$*:* $\mathbf{P}_{\mathrm{s}}(G, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\mathrm{s}}(\mathcal{P}'))$.

Definition 2.5. *The* Λ-sustained space complexity of $\mathcal{P}$ *is:* $\mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \Lambda) = |\{P_i : |P_i| \geq \Lambda\}|$. *The* Λ-sustained space complexity of G *is the minimal* Λ*-sustained space complexity of any valid pebbling strategy that pebbles the target set* $T \subseteq V$*:* $\mathbf{P}_{\mathrm{ss}}(G, \Lambda, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\mathrm{ss}}(\mathcal{P}', \Lambda))$.

Definition 2.6. *The* graph-optimal sustained complexity of $\mathcal{P}$ *is:*
$\mathbf{P}_{\text{opt-ss}}(\mathcal{P}) = \mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \mathbf{P}_{\mathrm{s}}(G, T))$. *The* graph-optimal sustained complexity of G *is the minimal graph-optimal sustained complexity of any valid pebbling strategy that pebbles the target set* $T \subseteq V$*:* $\mathbf{P}_{\text{opt-ss}}(G, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}'))$.

Definition 2.7. *The* Δ-suboptimal sustained complexity of $\mathcal{P}$ *is:*

$$\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Delta) = \mathbf{P}_{\mathrm{ss}}(\mathcal{P}, \mathbf{P}_{\mathrm{s}}(G, T) - \Delta).$$

The Δ-suboptimal sustained complexity of G *is the minimal graph-optimal sustained complexity of any valid pebbling strategy that pebbles the target set* $T \subseteq V$*:* $\mathbf{P}_{\text{opt-ss}}(G, \Delta, T) = \min_{\mathcal{P}' \in \mathbb{P}_{G,T}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}', \Delta))$.

2.2.2 Time complexity in standard pebbling

We present the following formal definitions for measuring the time complexity of strategies in the standard pebble game. In all the below definitions, $G = (V, E)$ is a graph, $T \subseteq V$ is a target set, $\mathcal{P} = (P_1, \ldots, P_t)$ is a standard pebbling strategy on (G, T) where $\mathbb{P}_{G,T,S}$ denotes the set of all valid pebbling strategies on (G, T) that use at most S pebbles.

Definition 2.8. *The* time complexity *of a pebbling strategy* $\mathcal{P}$ *is* $\mathsf{Time}(\mathcal{P}) = |\mathcal{P}|$. *The* time complexity *of a graph* $G = (V, E)$ *given that at most* S *pebbles can be used is* $\mathsf{Time}(G, S) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,S}} (\mathsf{Time}(\mathcal{P}))$.

2.2.3 Space complexity in black-magic pebbling

Next, we define the corresponding complexity notions for the black-magic pebbling game. As above, $G = (V, E)$ is a graph, $T \subseteq V$ is a target set, and $\mathfrak{M}$ is a magic pebble bound. In this subsection, $\mathcal{P} = (P_1, \ldots, P_t) = ((M_1, B_1), \ldots, (M_t, B_t))$ denotes a black-magic pebbling strategy on (G, T). Moreover, $\mathbb{M}_{G,T,\mathfrak{M}}$ denotes the set of all valid magic pebbling strategies on (G, T), and $m(\mathcal{P})$ denotes the total number of magic pebbles used in the execution of $\mathcal{P}$.

Definition 2.9. *The* (magic) space complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\mathrm{s}}(\mathcal{P}) = \max(m(\mathcal{P}), \max_{P_i \in \mathcal{P}} (|P_i|))$. *The* (magic) space complexity of G w.r.t. $\mathfrak{M}$ *is the minimal space complexity of any valid magic pebbling strategy that pebbles the target set* $T \subseteq V$*:* $\mathbf{P}_{\mathrm{s}}(G, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\mathrm{s}}(\mathcal{P}))$.

Remark. We briefly provide some intuition for the complexity measure defined above in Def. 2.9. If we consider all magic pebbles to be static memory objects that were saved from a previous evaluation of the hash function, then the total number of magic pebbles is the amount of memory that was used to save the results of a previous evaluation of the hash function. Because of this, it is natural to take the maximum of the memory used to store results from a previous evaluation of the function and the current memory that is used by our current pebbling strategy since that would represent how much memory was used to compute the results of hash function during the current evaluation.

Definition 2.10. *The* (magic) Λ-sustained space complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\text{ss}}(\mathcal{P}, \Lambda) = |\{P_i : |P_i| \geq \Lambda\}|$. *The* Λ-sustained space complexity of G w.r.t. $\mathfrak{M}$ and $T \subseteq V$ *is:* $\mathbf{P}_{\text{ss}}(G, \Lambda, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Lambda))$.

Definition 2.11. *The* (magic) graph-optimal sustained complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\text{opt-ss}}(\mathcal{P}) = \mathbf{P}_{\text{ss}}(\mathcal{P}, \mathbf{P}_{\text{s}}(G, T))$. *The* graph-optimal sustained complexity of G w.r.t. $\mathfrak{M}$ and $T \subseteq V$ *is:* $\mathbf{P}_{\text{opt-ss}}(G, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}))$.

Definition 2.12. *The* (magic) Δ-suboptimal sustained complexity *of* $\mathcal{P}$ *is:* $\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Delta) = \mathbf{P}_{\text{ss}}(\mathcal{P}, \mathbf{P}_{\text{s}}(G, T) - \Delta)$. *The* Δ-suboptimal sustained complexity of G w.r.t. $\mathfrak{M}$ and $T \subseteq V$ *is:*

$$\mathbf{P}_{\text{opt-ss}}(G, \Delta, \mathfrak{M}, T) = \min_{\mathcal{P} \in \mathbb{P}_{G,T,\mathfrak{M}}} (\mathbf{P}_{\text{opt-ss}}(\mathcal{P}, \Delta)).$$

2.3 Incrementally hard graphs

We introduce the following definition for our notion of graphs which require $|T|$ pebbles to pebble regardless of the number of targets that are asked, given a constraint on the number of magic pebbles that can be used. This concept has not been previously analyzed in the pebbling literature; traditional pebbling complexity usually treats graphs with fixed target sets.

Definition 2.13 (Incremental Hardness)**.** *Given at most* $\mathfrak{M}$ *magic pebbles, for any subset of targets* $C \subseteq T$ *where* $|C| > \mathfrak{M}$*, the number of pebbles (magic and black pebbles) necessary in the black-magic pebble game to pebble* C *is at least* $|T|$ *where the number of magic pebbles used in this game is upper bounded by* $\mathfrak{M}$*:* $\mathbf{P}_{\text{s}}(G, |C| - 1, C) \geq |T|$.

2.3.1 α-tradeoff cumulative complexity

α-tradeoff cumulative complexity, or CC^{α}, is a new measure introduced in this paper, which accounts for situations where space and time do not trade off linearly. Similar notions to this have been explored before e.g. [FLW13], [BK15, AB16, RD17]. A discuss of the *core-area memory ratio* [BK15, AB16, RD17] can be found in Section 1.2. They considered the notion of λ-memory-hardness where intuitively $S \cdot T = \Omega\left(G^{\lambda+1}\right)$ where the space-time cost is some exponential of the size of the stored graph [FLW13]. We note that this notion is very different from our notion of α-tradeoff complexity since they only consider the space-time cost (not cumulative complexity) and do not consider nonlinear tradeoffs between space and time (one can just consider $G^{\lambda+1}$ to a constant in the tradeoff curve).

Here, we see the usefulness of defining sustained complexities in terms of the minimum required space (as opposed to being parametrized by Λ) since we can always obtain an upper bound on CC^{α}, for *any* α, of a graph directly from our proofs of the space complexity and sustained time complexity of a DAG.

Definition 2.14 (Standard pebbling α-space cumulative complexity)**.** *Given a valid parallel standard pebbling strategy, $\mathcal{P}$, for pebbling a graph $G = (V, E)$, the* standard pebbling α-space cumulative complexity *is the following:*

$$\mathsf{p\text{-}cc}_\alpha(G, \mathcal{P}) = \sum_{P_i \in \mathcal{P}} |P_i|^\alpha \ .$$

Definition 2.15 (Black-magic pebbling α-space cumulative complexity)**.** *Given a valid parallel black-magic pebbling strategy, $\mathcal{P}$, for pebbling a graph $G = (V, E)$, the* black-magic pebbling α-space cumulative complexity *is the following:*

$$\mathsf{p\text{-}cc}^M_\alpha(G, \mathcal{P}) = \max\left(m(\mathcal{P})^\alpha, \sum_{P_i \in \mathcal{P}} |P_i|^\alpha\right) = \max\left(m(\mathcal{P})^\alpha, \sum_{P_i \in \mathcal{P}} |B_i \cup M_i|^\alpha\right)$$

where $m(\mathcal{P})$ denotes the total number of magic pebbles used in the magic pebbling strategy $\mathcal{P}$.

The following definition, CC^α, is an analogous definition to CC as defined by [AS15] (specifically, CC^α when $\alpha = 1$ is equivalent to CC) to account for varying costs of memory usage vs. time.

Definition 2.16 (CC^α)**.** *Given a graph, $G \in \mathbb{G}$, and a valid standard/magic pebbling strategy, $\mathcal{P}$, we define the $CC^\alpha(G)$ to be*

$$CC^\alpha(\mathcal{P}) = \left(\mathsf{p\text{-}cc}_\alpha\left(G, \mathcal{P}\right)\right).$$

Given a graph, $G \in \mathbb{G}$, and a family of valid standard pebbling strategies, $\mathbb{P}$, we define the $CC^\alpha(G)$ to be

$$CC^\alpha(G) = \min_{\mathcal{P} \in \mathbb{P}}\left(\mathsf{p\text{-}cc}_\alpha\left(G, \mathcal{P}\right)\right) \ ,$$

and, given a family $\mathbb{P}^M$ of valid black-magic pebbling strategies, we define $CC^\alpha(G)$ to be

$$CC^\alpha(G) = \min_{\mathcal{P}^M \in \mathbb{P}^M}\left(\mathsf{p\text{-}cc}^M_\alpha\left(G, \mathcal{P}^M\right)\right) \ .$$

3 Parallel random oracle model (PROM)

In this paper, we consider two broad categories of computations: *pebbling strategies* and *PROM algorithms*. Specifically, we discussed above the pebbling models and pebble games we use to construct our static memory-hard functions. Now, we define our PROM algorithms.

Prior work has observed the close connections between these two types of computations as applied to DAGs, and our work brings out yet more connections between the two models. In this section, we give an overview of how PROM computations work and define the complexity measures that we apply to PROM algorithms. Some of the complexity measures were introduced by prior work, and others are new in this work.

3.1 Overview of PROM computation

The random oracle model was introduced by [BR93]. When we say random oracle, we always mean a *parallel* random oracle unless otherwise specified.

An *algorithm* in the PROM is a probabilistic algorithm $\mathcal{B}$ which has parallel access to a stateless oracle $\mathcal{O}$: that is, $\mathcal{B}$ may submit many queries in parallel to $\mathcal{O}$. We assume $\mathcal{O}$ is sampled uniformly from an oracle set $\mathbb{O}$ and that $\mathcal{B}$ may depend on $\mathbb{O}$ but not $\mathcal{O}$.

The algorithm proceeds in discrete time-steps called *iterations*, and may be thought to consist of a series of algorithms $(\mathcal{B}_i)_{i\in\mathbb{N}}$, indexed by the iteration i, where each $\mathcal{B}_i$ passes a *state* $\sigma_i \in \{0,1\}^*$ to its successor $\mathcal{B}_{i+1}$. σ_0 is defined to contain the input to the algorithm. We write $|\sigma_i|$ to denote the size, in bits, of σ_i. We write $[\![\sigma_i]\!]$ to denote $\frac{|\sigma_i|}{w}$, where w is the output length of the oracle $\mathcal{O}$. In other words, $[\![\sigma_i]\!]$ is the size of σ_i when counting in words of size w. In each iteration, the algorithm $\mathcal{B}_i$ may make a *batch* $\mathbf{q}_i = (q_{i,1}, \ldots, q_{i,|\mathbf{q}_i|})$ of queries, consisting of $|\mathbf{q}_i|$ individual queries to $\mathcal{O}$, and instantly receive back from the oracle the evaluations of $\mathcal{O}$ on the individual queries, i.e., $(\mathcal{O}(q_{i,1}), \ldots, \mathcal{O}(q_{i,|\mathbf{q}_i|}))$.

At the end of any iteration, $\mathcal{B}$ can append values to a special output register, and it can end the computation by appending a special terminate symbol $\perp$ on that register. When this happens, the contents y of the output register, excluding the trailing $\perp$, is considered to be the output of the computation. To denote the process of sampling an output, y, provided input x, we write $y \leftarrow \mathcal{B}^{\mathcal{O}}(x)$.

Definition 3.1 (Oracle functions). *An* oracle function *is a collection* $\mathfrak{f} = \{f^{\mathcal{O}} : D \to R\}_{\mathcal{O}\in\mathbb{O}}$ *of functions with domain D and outputs in R indexed by oracles $\mathcal{O} \in \mathbb{O}$.*

A family of oracle functions *is a set* $\mathcal{F} = \{\mathfrak{f}_\kappa : D_\kappa \to R_\kappa\}_{\kappa\in\mathbb{N}}$ *where each $\mathfrak{f}_\kappa$ is indexed by oracles from an oracle set $\mathbb{O}_\kappa : \{0,1\}^\kappa \to \{0,1\}^\kappa$ indexed by a security parameter κ.*[14]

Definition 3.2 (Memory complexity of PROM algorithms). *The* memory complexity of $\mathcal{B}(x;\rho)$ *(i.e., the memory complexity of $\mathcal{B}$ on input x and randomness ρ) is defined as:*

$$\mathsf{mem}_{\mathbb{O}}(\mathcal{B}, x, \rho) = \max_{i\in\mathbb{N}} \{[\![\sigma_i]\!]\} \ . \tag{1}$$

Definition 3.3 (Λ-sustained memory complexity of PROM algorithms). *The* Λ-sustained memory complexity of $\mathcal{B}(x;\rho)$ *is defined as:*

$$\mathsf{s\text{-}mem}_{\mathbb{O}}(\Lambda, \mathcal{B}, x, \rho) = |\{i \in \mathbb{N} : |\sigma_i| \geq \Lambda\}| \ . \tag{2}$$

Note that (1) and (2) are distributions over the choice of $\mathcal{O} \leftarrow \mathbb{O}$.

3.2 Functions defined by DAGs

We now describe how to translate a graph construction into a function family, whose evaluation involves a series of oracle calls in the PROM. Any family of DAGs induces a family of *oracle functions* in the PROM, whose complexity is related to the pebbling complexity of the DAG. We first define the syntax of *labeling* of DAG nodes, then define a *graph function family.*

Definition 3.4 (Labeling). *Let $G = (V,E)$ be a DAG with maximum in-degree δ, let $\mathfrak{L}$ be an arbitrary "label set," and define $\mathbb{O}(\delta,\mathfrak{L}) = \left(V \times \bigcup_{\delta'=1}^{\delta} \mathfrak{L}^{\delta'} \to \mathfrak{L}\right)$. For any function $\mathcal{O} \in \mathbb{O}(\delta,\mathfrak{L})$ and any label $\zeta \in \mathfrak{L}$, the $(\mathcal{O},\zeta)$-labeling of G is a mapping $\mathsf{label}_{\mathcal{O},\zeta} : V \to \mathfrak{L}$ defined recursively as follows.*[15]

$$\mathsf{label}_{\mathcal{O},\zeta}(v) = \begin{cases} \mathcal{O}(v,\zeta) & \textit{if } \mathsf{indeg}(v) = 0 \\ \mathcal{O}(v, \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{pred}(v))) & \textit{if } \mathsf{indeg}(v) > 0 \end{cases} .$$

[14]For simplicity, we have the input and output domains of the oracles equal to $\{0,1\}^\kappa$, but this is not a necessary restriction: the sizes could be any polynomials in κ.

[15]We abuse notation slightly and also invoke $\mathsf{label}_{\mathcal{O},\zeta}$ on *sets* of vertices, in which case the output is defined to be a tuple containing the labels of all the input vertices, arranged in lexicographic order of vertices.

Definition 3.5 (Graph function family). *Let $n = n(\kappa)$ and let $\mathbb{G}_\delta = \{G_{n,\delta} = (V_n, E_n)\}_{\kappa \in \mathbb{N}}$ be a graph family. We write $\mathbb{O}_{\delta,\kappa}$ to denote the set $\mathbb{O}(\delta, \{0,1\}^\kappa)$ as defined in Definition A.1. The* graph function family *of $\mathbb{G}$ is the family of oracle functions $\mathcal{F}_\mathbb{G} = \{\mathfrak{f}_G\}_{\kappa \in \mathbb{N}}$ where $\mathfrak{f}_G = \{f_G^\mathcal{O} : \{0,1\}^\kappa \to (\{0,1\}^\kappa)^z\}_{\mathcal{O} \in \mathbb{O}_{\delta,\kappa}}$ and $z = z(\kappa)$ is the number of sink nodes in G. The output of $f_G^\mathcal{O}$ on input label $\zeta \in \{0,1\}^\kappa$ is defined to be*

$$f_G^\mathcal{O}(\zeta) = \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{sink}(G)) \ ,$$

where $\mathsf{sink}(G)$ is the set of sink nodes of G.

3.3 Relating complexity of PROM algorithms and pebbling strategies

Any PROM algorithm $\mathcal{B}$ and input x induce a black-magic pebbling strategy, $\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$, called an *ex-post-facto black-magic pebbling strategy.* The way in which this strategy is induced is similar to *ex-post-facto pebbling* as originally defined by [AS15] in the context of the standard pebble game. We adapt their technique for the black-magic game.

Definition 3.6 (Ex-post-facto black-magic pebbling). *Let $n = n(\kappa)$ and let $\mathbb{G}_\delta = \{G_{n,\delta} = (V_n, E_n)\}_{\kappa \in \mathbb{N}}$ be a graph family. Let $\zeta = \zeta(\kappa) \in \{0,1\}^\kappa$ be an arbitrary input label for the graph function family $\mathcal{F}_\mathbb{G}$. For any $v \in V_n$, define*

$$\mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v) = (v, \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{pred}(v))) \ .$$

Let $\mathcal{B}$ be a non-uniform PROM algorithm. Fix an implicit security parameter κ. Let x be an input to $\mathcal{B}$. We now define a magic pebbling strategy induced by any given execution of $\mathcal{B}^\mathcal{O}(x; \$)$, where \$ denotes the random coins of $\mathcal{B}$. Such an execution makes a sequence of batches of random oracle calls (as defined in Section 3.1), which we denote by

$$\mathbf{q}(\mathcal{B}, \mathcal{O}, x, \$) = (\mathbf{q}_1, \ldots, \mathbf{q}_t) \ .$$

The induced black-magic pebbling strategy,

$$\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$) = ((B_0, M_0), \ldots, (B_t, M_t)) \ , \tag{3}$$

is called an ex-post-facto black-magic pebbling*, and is defined by the following procedure.*

```
1. $B_0 = M_0 = \varnothing$.
2. For $i = 1, \ldots, t$:
   (a) $B_i = B_{i-1}$.
   (b) $M_i = M_{i-1}$.
   (c) For each individual query $q \in \mathbf{q}_i$, if there is some $v \in V_n$ such that $q = \mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v)$ and
       $v \notin P_i$, then "pebble $v$" by performing the following steps:
       i. If $\mathsf{pred}(v) \subseteq M_i \cup B_i$:
          • $B_i = B_i \cup \{v\}$.
       ii. Else:
          • $V = \{v\}$.
          • Let $V^*$ be the transitive closure of $V$ under the following operation:
            $V = V \cup \left(\bigcup_{v' \in V} \mathsf{pred}(v') \cap (M_i \cup B_i)\right)$.
          • $M_i = M_i \cup V^*$.
3. For $i = 1, \ldots, t$:
```

(a) *A node $v \in M_i \cup B_i$ is said to be* necessary at time i *if*

$$\exists j \in [t], q \in \mathbf{q}_j, v' \in V_n \ \text{s.t.} \quad j > i \wedge v \in \mathsf{pred}(v') \wedge q = \mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v') \\ \wedge \left(\nexists k \in [t], q' \in \mathbf{q}_k \ \text{s.t.}\ i < k < j \wedge q' = \mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(v) \right) .$$

In other words, a node is necessary if its label will be required in a future oracle call, but its label will not be obtained by any oracle query between now and that future oracle call. Remove from B_i and M_i all nodes that are not necessary at time i.

3.4 Legality and space usage of ex-post-facto black-magic pebbling

The following theorems establish that the space usage of PROM algorithms is closely related to the space usage of the induced pebbling.

We will use the following supporting lemma, also used in prior work such as [AS15, DKW11] (see, e.g., [DKW10] for a proof).

Lemma 3.7. *Let $B = b_1, \ldots, b_u$ be a sequence of random bits and let $\mathbb{H}$ be a set. Let $\mathcal{P}$ be a randomized procedure that gets a hint $h \in \mathbb{H}$, and can adaptively query any of the bits of B by submitting an index i and receiving b_i as a response. At the end of its execution, $\mathcal{P}$ outputs a subset $S \subseteq \{1, \ldots, u\}$ of $|S| = \varphi$ indices which were not previously queried, along with guesses for the values of the bits $\{b_i : i \in S\}$. Then the probability (over the choice of B and the randomness of $\mathcal{P}$) that there exists some $h \in \mathbb{H}$ such that $\mathcal{P}(h)$ outputs* all *correct guesses is at most $|\mathbb{H}|/2^{\varphi}$.*

Lemma 3.8 (Legality and magic pebble usage of ex-post-facto black-magic pebbling)**.** *Let $n = n(\kappa)$ and let $\mathbb{G}_\delta = \{G_{n,\delta} = (V_n, E_n)\}_{\kappa \in \mathbb{N}}$ be a graph family. Let $\zeta \in \{0,1\}^\kappa$ be an arbitrary input label for $\mathbb{G}_\delta$. Fix any efficient PROM algorithm $\mathcal{B}$ and input x. With overwhelming probability over the choice of random oracle $\mathcal{O} \leftarrow \mathbb{O}$ and the random coins \$ of $\mathcal{B}$, it holds that the ex-post-facto magic pebbling $\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$ consists of valid magic-pebbling moves, and uses fewer than $\chi = \left\lfloor \frac{|x|}{\kappa - \log(q)} + 1 \right\rfloor$ magic pebbles (i.e., for all i, $|M_i| \leq \chi$), where q is the number of oracle queries made by $\mathcal{B}(x)$.*

Proof. Fix an algorithm $\mathcal{B}$ and, for the sake of contradiction, suppose that there is an input x such that with non-negligible probability over $\mathcal{O}$ and \$, the induced pebbling $\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$ uses at least χ magic pebbles or contains an invalid move. By definition, this means that the following event $\mathcal{E}$ occurs with non-negligible probability: on at least χ occasions, a (magic) pebble is placed on a node v although its parents were not all pebbled in the previous step. In turn, this means that a correct random-oracle query for the label of v is made by $\mathcal{B}$; and the correct query contains the label of some predecessor node v' which was not contained in the output of any previous oracle call.

Let us suppose that event $\mathcal{E}$ occurs with probability more than $p = \frac{q^\chi 2^{|x|}}{2^{\kappa\chi}}$. Note that this probability is negligible, since

$$\begin{aligned} p = \frac{q^\chi 2^{|x|}}{2^{\kappa\chi}} &= 2^{\chi \log(q) + |x| - \kappa\chi} \\ \chi \log(q) + |x| - \kappa\chi &= \chi(\log(q) - \kappa) + |x| && \text{(analyzing the exponent)} \\ &= \left\lfloor \frac{|x|}{\kappa - \log(q)} + 1 \right\rfloor (\log(q) - \kappa) + |x| && \text{(substituting for } \chi\text{)} \\ &\leq \frac{|x| + \kappa - \log(q)}{\kappa - \log(q)} (\log(q) - \kappa) + |x| \\ &= -(|x| + \kappa - \log(q)) + |x| && \text{(canceling denominator)} \\ &= -\kappa + \log(q) \end{aligned}$$

and q is polynomial in κ. Based on this assumption, we construct a predictor that predicts χ output values of the random oracle with impossibly high probability (specifically, violating Lemma 3.7) as follows. The predictor $\mathcal{P}$ depends on input x and can query the random oracle on inputs of its choice, before outputting its prediction. Let $\hat{r}$ be an upper bound on the number of random bits used by $\mathcal{B}(x)$. The predictor also has access to a sequence $\hat{R}$ of $\hat{r}$ random bits, that it can use to simulate the random coins of $\mathcal{B}$.

- *Hint:* The predictor $\mathcal{P}$ receives as its hint[16] *either* $\perp$ if the induced pebbling $\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$ is valid and uses no more than χ magic pebbles, *or* the following information otherwise:
  - the index $i^* \in [q]$ of the first oracle call causing the illegal event (inducing the χth placement of a magic pebble on some node v) to happen;
  - the indices $I \subset [i^*]$ of all oracle calls preceding the i^*th oracle call, that induce the placement of a magic pebble or pebbles; and
  - $\mathcal{B}$'s input x.

  The size of this hint is at most $\chi \log(q) + |x|$ bits.
- *Execution:* If the hint is $\perp$, then $\mathcal{P}$ halts and outputs nothing. Otherwise, $\mathcal{P}$ runs $\mathcal{B}(x; \hat{R})$, forwarding all oracle calls to the random oracle, until the i^*th query. By construction, for each $i' \in I \cup \{i^*\}$, the i'th query contains the labels of the parents of the node $v_{i'}$ whose pebbling is induced by the i'th query, and at least one of these labels (say, label $\ell_{w_{i'}}$ for parent node $w_{i'}$) was not the output of any previous query to the random oracle. For each $i' \in I \cup \{i\}$, our predictor recomputes the value $\tilde{w}_{i'} = \mathsf{pre\text{-}lab}_{\mathcal{O},\zeta}(w_{i'})$ which is the preimage under $\mathcal{O}$ of $\ell_{w_{i'}}$. Note that by definition of $\mathsf{pre\text{-}lab}$, $\tilde{w}_{i'}$ can be computed without ever querying $\mathcal{O}$ on input $\tilde{w}_{i'}$. Finally, $\mathcal{P}$ outputs the following pairs:

  $$\left\{(\tilde{w}_{i'}, \ell_{w_{i'}})\right\}_{i' \in I \cup \{i^*\}} .$$

  Since by construction, each query $i' \in I \cup \{i^*\}$ induced the placement of a magic pebble, it follows that each pair $(\tilde{w}_{i'}, \ell_{w_{i'}})$ is a valid input-output pair of $\mathcal{O}$. Moreover, $\mathcal{P}$ never queried $\mathcal{O}$ on any $\tilde{w}_{i'}$.

The predictor's hint is $\perp$ with probability at most that of $\mathcal{E}$, and the predictor succeeds whenever the hint is not $\perp$. Hence, by our assumption about the probability p of the event $\mathcal{E}$, the predictor must succeed with probability greater than $p = \frac{q^\chi 2^{|x|}}{2^{\kappa\chi}}$. By construction, the size of the predictor's hint set is at most $q^\chi 2^{|x|}$, and the predictor's output is $\kappa\chi$ bits long. Thus Lemma 3.7 implies that the probability (over the choice of $\mathcal{O}$ and the randomness of $\mathcal{P}$) that there is some hint such that $\mathcal{P}$ outputs all correct guesses is at most $\frac{q^\chi 2^{|x|}}{2^{\kappa\chi}}$. (This is equal to p.) We have a contradiction, and the lemma follows. $\square$

Lemma 3.9 (Space usage of ex-post-facto black-magic pebbling)**.** *Let $n, \mathbb{G}_\delta, \zeta$ be as in Lemma 3.8. Fix any PROM algorithm $\mathcal{B}$ and input x. Fix any $i \in [t]$, $\lambda \geq 0$, and define*

$$\mathsf{epf\text{-}magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$) = (P_1^\mathcal{O}, \ldots, P_t^\mathcal{O}) = ((B_1^\mathcal{O}, M_1^\mathcal{O}), \ldots, (B_t^\mathcal{O}, M_t^\mathcal{O}))$$

for oracle $\mathcal{O}$. We may omit the superscript $\mathcal{O}$ for notational simplicity. It holds for all large enough κ that the following probability is overwhelming:

$$\Pr\left[\forall i \in [t],\ |P_i| \leq \chi'\right] ,$$

where $\chi' = \left\lfloor \frac{|\sigma_i|}{\kappa - \log(q)} + 1 \right\rfloor$, q is the number of oracle queries made by $\mathcal{B}$, and the probability is taken over $\mathcal{O} \leftarrow \mathbb{O}$ and the coins of $\mathcal{B}$.

[16]Note that the hint may depend both on the choice of random oracle, and on the randomness $\hat{R}$.

Proof. This proof has a very similar structure to that of Lemma 3.8. Assume for contradiction that with non-negligible probability for some $i \in [t]$ it holds that $|P_i| > \chi'$. Let $\mathcal{E}$ denote the event that the induced pebbling $\textsf{epf-magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$ satisfies $|P_i| > \chi'$, and suppose that $\mathcal{E}$ occurs with probability more than $p = \frac{q^{\chi'} 2^{|\sigma_i|}}{2^{\kappa\chi'}}$. Note that p is negligible, since

$$\begin{aligned}
p = \frac{q^{\chi'} 2^{|\sigma_i|}}{2^{\kappa\chi'}} &= 2^{\chi' \log(q) + |\sigma_i| - \kappa\chi'} \\
\chi' \log(q) + |\sigma_i| - \kappa\chi' &= \chi'(\log(q) - \kappa) + |\sigma_i| && \text{(analyzing the exponent)} \\
&= \left\lfloor \frac{|\sigma_i|}{\kappa - \log(q)} + 1 \right\rfloor (\log(q) - \kappa) + |\sigma_i| && \text{(substituting for } \chi'\text{)} \\
&\leq \frac{|\sigma_i| + \kappa - \log(q)}{\kappa - \log(q)} (\log(q) - \kappa) + |\sigma_i| \\
&= -(|\sigma_i| + \kappa - \log(q)) + |\sigma_i| && \text{(canceling denominator)} \\
&= -\kappa + \log(q)
\end{aligned}$$

We design a predictor $\mathcal{P}$ to predict the labels of all nodes in P_i with impossibly high probability, as follows. We refer to the oracle call that causes the ex-post-facto pebbling of a node $v \in P_i$ a *critical call.* (Critical calls encompass both black and magic pebble placements.) $\mathcal{P}$ depends on σ_i, $\mathcal{O}$, and a long enough sequence $\hat{R}$ of random bits used to simulate the coins of $\mathcal{B}$.

- *Hint:* The predictor $\mathcal{P}$ receives as its hint *either* $\perp$ if the induced pebbling $\textsf{epf-magic}_\zeta(\mathcal{B}, \mathcal{O}, x, \$)$ satisfies $|P_i| \leq \chi'$, *or* the following information otherwise:
  - the indices $J = \{j_1, \ldots, j_c\} \in [q]^{|P_i|}$ of the critical calls made by $\mathcal{B}$, and
  - the state σ_i outputted by $\mathcal{B}$ at the end of iteration i, and

  The size of this hint is $|P_i| \log(q) + |\sigma_i|$ bits. By our assumption on $|P_i|$, this is more than $\chi' \log(q) + |\sigma_i|$ bits.
- *Execution:* $\mathcal{P}$ runs $\mathcal{B}$ on input (z, 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 iff 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, $Q = \textsf{pre-lab}_{\mathcal{O},\zeta'}(w_{i'})$ and no new oracle calls need be made by the predictor to check this.
  - If the call is correct and the label of v has already been recorded then output the label. Otherwise query $\mathcal{O}$ to answer the call.

  Finally, $\mathcal{P}$ outputs predictions of all of the labels of the magic pebbles and all the labels associated with P_i, as follows.
  - The labels of the magic pebbles are determined as described in the proof of Lemma 3.8.
  - When $\mathcal{B}$ terminates, $\mathcal{P}$ checks the transcript to determine the set B_i. It is easy to verify that their labels were never queried to $\mathcal{O}$ by $\mathcal{P}$. Then, for all $v \in B_i$ the predictor computes $\tilde{v} = \textsf{pre-lab}_{\mathcal{O},\zeta'}(v)$ and outputs the pair $(\tilde{v}, \ell_v)$ where ℓ_v is the label of v (as specified in the input of the oracle call for associated critical call).

The predictor's hint is $\perp$ with probability at most that of $\mathcal{E}$, and the predictor succeeds whenever the hint is not $\perp$. Hence, by our assumption about the probability p of the event $\mathcal{E}$, the predictor must succeed with probability greater than p. The predictor's output is $\kappa|P_i| > \kappa\chi'$ bits long. From Lemma 3.7, it follows that the probability (over the choice of $\mathcal{O}$ and the randomness of $\mathcal{P}$) that there is some hint such that $\mathcal{P}$ outputs all correct guesses is at most $(q^{\chi'} 2^{|\sigma_i|})/2^{\kappa\chi'}$. (This is equal to p.) We have a contradiction and the lemma follows. □

4 Static-memory-hard functions

We now define *static-memory-hard functions.* As mentioned above, prior notions of memory-hardness consider only dynamic memory usage. To model static memory usage, we consider a hash function with two parts $(\mathcal{H}_1, \mathcal{H}_2)$ where $\mathcal{H}_2(x)$ computes the output of the hash function $h(x)$ given oracle access to the output of $\mathcal{H}_1$. This design can be seen to reduce honest party computation time by limiting the hard work to one-off preprocessing phase, while maintaining a large space requirement for password-cracking adversaries. Informally, our guarantee says that unless the adversary stores a specified amount of *static* memory, he must use an equivalent amount of *dynamic* memory to compute h correctly on many outputs. Definition 4.1 is syntactic and Definition 4.2 formalizes the memory-hardness guarantee.

Notation PPT stands for "probabilistic polynomial time." For $\vec{b} \in \{0,1\}^*$, define $\mathsf{Seek}_{\vec{b}} : \{1, \ldots, |\vec{b}|\} \to \{0,1\}$ to be an oracle that on input ι returns the ιth bit of $\vec{b}$.

Definition 4.1 (Static-memory hash function family (SHF))**.** *A static-memory hash function family $\mathcal{H}^{\mathcal{O}} = \{h^{\mathcal{O}}_{\kappa} : \{0,1\}^{w'} \to \{0,1\}^{w}\}_{\kappa \in \mathbb{N}}$ mapping $w' = w'(\kappa)$ bits to $w = w(\kappa)$ bits is described by a pair of deterministic oracle algorithms $(\mathcal{H}_1, \mathcal{H}_2)$ such that for all $\kappa \in \mathbb{N}$ and $x \in \{0,1\}^n$,*

$$\mathcal{H}_2^{\mathsf{Seek}_{\hat{R}}}(1^{\kappa}, x) = h_{\kappa}(x), \ \ \textit{where } R = \mathcal{H}_1(1^{\kappa}) \ .$$

(The superscript $\mathcal{O}$ is left implicit.)

The next definition presents a parametrized notion of $(\Lambda, \Delta, \tau, q)$-hardness of an SHF. Before delving into the formal definition, we give a brief intuition of the guarantee provided by Definition 4.2: any adversary who produces at least q correct input-output pairs of the hash function must *either* have used $\Lambda - \Delta$ static memory *or* incur a requirement of Λ dynamic memory *sustained over* τ time-steps at runtime.

The role of q. The parameter q in Definition 4.2 serves to capture the intuitive idea that an adversary that uses a certain amount of space could always use that space to directly store output values of h_κ. Clearly, an adversary with an arbitrary input R could very easily output up to ▯$|R|$▯ correct output values. Our goal is to lower bound the amount of space needed by an adversary who outputs nontrivially more correct values than that — and q, which is a function of $|R|$, captures how many more.

Definition 4.2 ($(\Lambda, \Delta, \tau, q)$-hardness of SHF)**.** *Let $\mathcal{H} = \{h_\kappa\}_{\kappa \in \mathbb{N}}$ be a static-memory hash function family described by algorithms $(\mathcal{H}_1, \mathcal{H}_2)$, mapping w' to w bits. $\mathcal{H}$ is (Λ, Δ, τ)-hard if for any large enough $\kappa \in \mathbb{N}$, any string $R \in \{0,1\}^{\Lambda - \Delta}$, and any PPT algorithm $\mathcal{A}$, for any set $X = \{x_1, \ldots, x_q\} \subseteq \{0,1\}^{w'}$, there is a negligible ε such that*

$$\Pr_{\mathcal{O}, \rho}\Big[\big\{(x_1, h_\kappa(x_1)), \ldots, (x_q, h_\kappa(x_q))\big\} = \mathcal{A}(1^\kappa, R; \rho) \wedge \mathsf{s\text{-}mem}_{\mathbb{O}}(\Lambda, \mathcal{A}, R, \rho) < \tau\Big] < \varepsilon \ .$$

For simplicity, we henceforth assume $w' = w = \kappa$ (i.e., the oracle's input and output sizes are equal to the security parameter) unless otherwise stated.

4.1 Dynamic SHFs

As discussed in detail in the introduction, static memory requirements are orthogonal and complementary to dynamic memory requirements of MHFs as formalized by [AS15]. Given a pebbling-based SHF and a pebbling-based MHF, they can be combined by simple concatenation into a "dynamic

SHF," a function that inherits both the static memory requirement of the former and the dynamic memory requirement of the latter, as outlined (informally) next.

Let $\mathcal{H}_{\mathsf{dyn}}^{\mathcal{O}}$ be a dynamic MHF and $(\mathcal{H}_1^{\mathcal{O}}, \mathcal{H}_2^{\mathcal{O}})$ be a SHF family, and the computation of both of these correspond to computing labels of nodes in a DAG as a function of a pebbling algorithm and a random oracle $\mathcal{O}$. We construct a dynamic SHF $\mathcal{H}^{\mathcal{O}}$ that is defined as follows: on input $(1^\kappa, x)$, output $\mathcal{H}_2^{\mathcal{O}(0,\cdot)}(1^\kappa, x)||\mathcal{H}_{\mathsf{dyn}}^{\mathcal{O}(1,\cdot)}(1^\kappa, x)$. The resulting $\mathcal{H}^{\mathcal{O}}$ inherits both the MHF guarantees of $\mathcal{H}_{\mathsf{dyn}}$ and the SHF guarantees of $(\mathcal{H}_1, \mathcal{H}_2)$. Note that importantly, the labels of the nodes in the graphs corresponding to the MHF $\mathcal{H}_{\mathsf{dyn}}^{\mathcal{O}(0,\cdot)}$ and the SHF $(\mathcal{H}_1^{\mathcal{O}(1,\cdot)}, \mathcal{H}_2^{\mathcal{O}(1,\cdot)})$ are independent as the MHF and the SHF use disjoint partitions of the random oracle domain.

Using this method, our SHF constructions can be combined with existing MHF constructions such as [AS15], [ABP17a], [ABP17b], yielding a "best of both worlds" dynamic SHF that enjoys both types of memory-hardness.

5 SHF constructions

A first attempt What if we pebble a hard-to-pebble graph, and then let $R_{k,i} = H(P(k), i)$ where $P(k)$ is the entire pebbling of the graph (on input k and iteration i is the i-th call to the hash function H)? This would in fact work in the random oracle model where the random oracle takes arbitrary-length input. However, in practice, hash functions do not take arbitrary-length input. While constructions like Merkle-Damgård [Mer79] and sponge [BDPA08] can transform a fixed-input-length hash function into one that takes arbitrary-length inputs, the resulting function does *not* behave like a random oracle even if the fixed-length hash function does.[17] Moreover, the computation graphs of known length-expanding transformations such as Merkle-Damgård and sponge functions require very little space to compute. For instance, the computation graph of the Merkle-Damgård construction is a binary tree and the computation graph of the sponge function is a caterpillar graph both of which take logarithmic and constant space, respectively, to compute. Thus, we have to use special constructions to achieve the local-hardness properties we need.

Recall from Definition 2.13 that the property we want is this "locally hard to access" notion, meaning that if an adversarial party chooses to not store the static part of our hash function which they obtain from performing the "preprocessing" computation associated with $\mathcal{H}_1$, then they must use the same memory and sustained time to recompute the function when our static-memory-hard function is called on *any subset of inputs* larger than the memory used to store the preprocessed computation. We achieve this desired property in our $\mathcal{H}_1$ functions using two novel DAG constructions, one of which is optimal for a specific graph class and the other we conjecture to be optimal for all general graph classes.

5.1 $\mathcal{H}_1$ constructions

We first note the differences between the graph constructions we present here and the constructions presented in previous literature [AS15, ACK$^+$16, ABP17a, DFKP15]. Firstly, many of the constructions presented in previous work feature a single target node. This is reasonable in the context of memory-hard functions since both the honest party and the adversary must compute the hash function dynamically (obtaining a single label as the output of the function) on each input. However, in our context of static-memory-hard functions, single-target-node constructions do not

[17]For example, both the constructions mentioned process the input sequentially in chunks. Evaluating the hash function on inputs that differ only in the final chunk will yield outputs that differ in a known way; this provides a way to distinguish these constructions from a random oracle even if the underlying fixed-length hash function is a random oracle.

make sense. Secondly, our constructions differ from even the multiple target node constructions presented in the literature (specifically, the constructions of [DFKP15]) since prior constructions mainly focused on finding graphs that have large memory vs. time tradeoffs.

Our constructions are designed with the goal that any adversary that does not store almost all the target labels must dynamically use *the same amount of space as needed to store all the labels* to compute the hash function (while *still incurring a cost in runtime*). Moreover, our constructions based on local hardness ensure a stronger guarantee than the constructions in [DFKP15]; in our case, one must use at least S space (for some definition of S) to compute *any* given subset of targets larger than one's current memory usage, whereas in their case, they use S space to compute some subset of targets chosen uniformly at random. Therefore, our specifications are stronger in that we provide a space bound as well as a time bound for adversaries; and moreover, for *honest* parties, the time cost is only a one-time setup cost. We prove our pebbling costs in terms of the black-magic pebble game (defined in Section 2) as opposed to the standard pebble game used in previous works. Most notably, this means that in all of our constructions, the pebbling number is upper bounded by the number of targets (since one can always just pebble the targets with magic pebbles).

We begin with some simple and clean constructions of $\mathcal{H}_1$ based on pebbling constructions that exist in the literature. We first prove a lemma regarding the minimum number of pebbles used in the PROM model and the minimum number of pebbles used in the sequential memory model. This is useful in more than one way: (1) it tells us that parallelization does not save the adversary in space so honest parties (who can only compute a constant number of labels at a time) and adversaries (who can compute an arbitrary number of labels at the same time) operate under the same space constraints and (2) it allows us to directly compare sustained time complexities between adversaries and honest parties with respect to space usage .

Lemma 5.1 (Standard Pebbling Sequential/Parallel Equivalence)**.** *Given a DAG $G = (V, E)$, $\mathbf{P}_{\mathrm{s}}(G, T) = \mathbf{P}_{\mathrm{s}}^{||}(G, T)$ where $\mathbf{P}_{\mathrm{s}}(G, T)$ is defined to be the minimum standard pebbling space complexity in the sequential model, and we define $\mathbf{P}_{\mathrm{s}}^{||}(G, T)$ to be the minimum standard pebbling space complexity in the parallel model.*

Proof. Any sequential pebbling strategy, $\mathcal{P}$ can be simulated by a parallel pebbling strategy, $\mathcal{P}^{||}$ since $\mathcal{P}^{||}$ can choose to place one pebble at a time. Therefore, $\mathbf{P}_{\mathrm{s}}^{||}(G, T) \leq \mathbf{P}_{\mathrm{s}}(G, T)$. We now show that there exists a sequential pebbling strategy, $\mathcal{P}$, that uses the same number of pebbles to pebble a graph as a parallel strategy $\mathcal{P}^{||}$. Suppose that at time i, a set of pebbles are added to nodes in P_i in G under algorithm $\mathcal{P}^{||}$. Then, $pred(P_i)$ must be pebbled at time $i - 1$. $\mathcal{P}$ can thus spend $|P_i \backslash P_{i-1}|$ pebbling steps to pebble the graph sequentially by adding pebbles on all vertices $v \in P_i \backslash P_{i-1}$ sequentially until the state of the graph is the same as the state of the graph at time i under strategy $\mathcal{P}^{||}$. Similarly, if a set of pebbles D_i are deleted from the graph at time i, then $\mathcal{P}$ can choose to spend at most $|D_i|$ sequential pebbling steps to delete $|D_i|$ pebbles. If both strategies start on identical graphs with the same starting configuration P_0, then we have shown that $\mathbf{P}_{\mathrm{s}}^{||}(G, T) \geq \mathbf{P}_{\mathrm{s}}(G, T)$. Thus, $\mathbf{P}_{\mathrm{s}}^{||}(G, T) = \mathbf{P}_{\mathrm{s}}(G, T)$. $\square$

We use Lemma 5.1 to prove an equivalent lemma for the black-magic pebble game below.

Lemma 5.2 (Black-Magic Pebbling Sequential/Parallel Equivalence)**.** *Given a DAG $G = (V, E)$, $\mathbf{P}_{\mathrm{s}}(G, |T|, T) = \mathbf{P}_{\mathrm{s}}{}^{||}(G, |T|, T)$ where $\mathbf{P}_{\mathrm{s}}(G, |T|, T)$ was defined to be the minimum black-magic pebbling space complexity in the sequential model, and we define $\mathbf{P}_{\mathrm{s}}{}^{||}(G, |T|, T)$ to be the minimum black-magic pebbling space complexity in the parallel model.*

Proof. Any placement of black pebbles can be translated from the sequential to the parallel pebbling strategy and vice versa using the techniques stated in the proof of Lemma 5.1. Any sequential pebbling placement of magic pebbles can be simulated trivially by a parallel pebbling strategy. Any parallel pebbling placement of M magic pebbles can be simulated via a sequential pebbling strategy using M additional steps. Thus, $\mathbf{P}_{\mathrm{s}}(G, |T|, T) = \mathbf{P}_{\mathrm{s}}^{\|}(G, |T|, T)$. □

Now, we jump into our constructions. We first provide a simple construction and show why this construction is not optimal. In addition, we define some subgraph components in the pebbling literature that are important subcomponents of our constructions.

5.1.1 A failed attempt at $\mathcal{H}_1$

We first provide a failed attempt at constructing $\mathcal{H}_1$ due to the large amount of time that is needed to compute the function (for the sequential honest party) with respect to the amount of memory needed to store the output of the function. In other words, this construction is problematic in the sense that an exponential number of steps is necessary to compute the stored results of the function from scratch for the honest party but the adversary with parallel processing time can compute the function from scratch in linear time. Although the honest party could obtain the results of the preprocessing (i.e. the static part of the hash function) from elsewhere, we must ensure that they can still feasibly compute $\mathcal{H}_1$ themselves in the event that they do not trust any of the sources from which they can obtain the static data.

Intuitively, our failed attempt at constructing $\mathcal{H}_1$ is a series of binary search trees. From here onwards, we describe all constructions of $\mathcal{H}_1$ as a directed acyclic graph with n nodes and later use our theorems above to prove static memory hardness from our constructed DAGs.

Graph Construction 5.3 (Composite Binary Tree DAG). *Let B_h^C be a composite binary tree DAG with height h constructed in the following way where T is the number of targets of our DAG. Let $s = |T|$. In our intended construction $h = s$.*

1. *Let the set of nodes be V. Let the set of edges be E.*
2. *Create $(s+1)2^{h-1} + s$ nodes.*
3. *Create $s+1$ binary search trees using $(s+1)2^{h-1}$ nodes in total where edges are directed from children to parents in each binary tree. Let r_i for $i \in [1, s+1]$ be the roots of these binary search trees.*
4. *Order the remaining nodes in some arbitrary order, let s_j be the jth node in this order for $j \in [1, s]$.*
5. *Create directed edges (r_i, s_i) and $(r_{i+1 \bmod s}, s_i)$ for all $i \in [1, s]$.*

Given any binary search tree with height h, the minimum number of pebbles necessary to pebble the tree is h (assuming a 'tree' with one node has height 1) using the rules of the standard pebble game. Therefore, to ensure that the apex of the tree is pebbled and that both the honest party and the adversary both use h space to pebble the apex, the number of leaves necessary at the base of the tree is 2^{h-1}. If we suppose that the computationally weak honest party (who does not build special circuits) can only evaluate a constant number of random oracle calls at a time (place a constant number of pebbles), the number of sequential evaluations necessary for the honest party is $\geq \Omega(2^h)$ which is infeasible to accomplish. In constrast, the adversary only has to make $O(h)$ parallel random oracle calls, an exponential factor difference between the honest party and the adversary! Such a construction fails since it is clearly infeasible for the honest party since they would never be able to compute all target values of $\mathcal{H}_1$ from scratch (since this computation requires exponential time for the honest party). Thus, we would like a construction that has the same minimum space requirement but also small sequential evaluation time. We prove a better (but also simply defined) construction below.

5.1.2 Cylinder construction

We make use of what is defined in the pebbling literature as a *pyramid graph* [GLT80] in constructing our *cylinder graph*. The key characteristic of the pyramid graph we use is that the number of pebbles that is required to pebble the apex of the pyramid is equal to the height of the pyramid [GLT80] using the rules of the standard pebble game. Note that a pyramid by itself is not useful for our purposes since the black-magic pebbling space complexity of a pyramid with one apex is 1. Therefore, we need to be able to use the pyramid in a different construction that uses superconstant number of pebbles in the magic pebble game in order to successfully pebble all target nodes.

Graph Construction 5.4 (Illustrated in Fig. 2)**.** *Let Π_h^C be a* cylinder *graph with height h. We define Π_h^C as follows:*

1. *Create $2h^2$ nodes. Let this set of $2h^2$ nodes be V.*
2. *Arrange the nodes in V into $2h$ levels of h nodes each, ranging from level 0 to level $2h-1$. Let the j-th node in level i be v_i^j. Create directed edges $(v_i^{j \bmod h}, v_{i+1}^{j \bmod h})$ and $(v_i^{j \bmod h}, v_{i+1}^{(j+1) \bmod h})$ for all $i \in [0, 2h-2]$. Let this set of edges be E.*

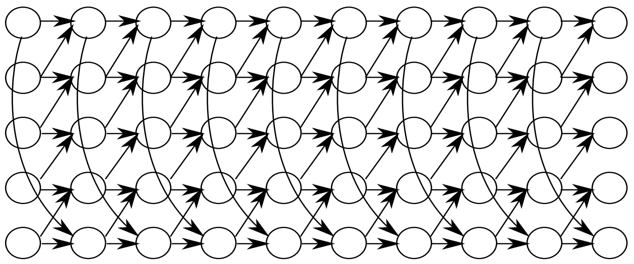


Figure 2: Cylinder construction (Def. 5.4) for $h = 5$.

Lemma 5.5. *Given a cylinder graph with height h, Π_h^C, $\mathbf{P}_{\text{s}}(\Pi_h^C, T) \geq h$.*

Proof. Let T be the target nodes of Π_h^C. Each target node is connected to a pyramid of height h. Therefore, by the proofs of minimum pebbling cost of pyramids given in [GLT80], the pyramid requires h pebbles to pebble using the rules of the standard pebble game. Therefore, to pebble any one target node $t \in T$ requires h pebbles, so pebbling all target nodes of Π_h^C, T, trivially requires h pebbles. $\square$

Lemma 5.6. $\mathbf{P}_{\text{opt-ss}}(\Pi_h^C, T) \geq 2h$.

Proof. The depth of Π_h^C is $2h$ (i.e. the longest directed path in Π_h^C has length $2h$). Thus, the minimum number of parallel steps necessary to pebble any $v \in T$ is $2h$. Let L_i be the set of nodes at the i-th level of Π_h^C where T is at level $2h-1$ and S is at level 0. To pebble each target node requires that all vertices in L_{h-1} (v_{h-1}^i for all $i \in [1,h]$) be pebbled at some time step simultaneously[18], $t \in [0, t_{\mathcal{P}}]$, by normality of pebbing strategies (see the definition of *frugal* and *normal* strategies in Definitions B.1 and B.2 [GLT80, DL17]). given any normal strategy $\mathcal{P}$. Thus, at least h parallel time steps where h pebbles are on the graph simultaneously are necessary to pebble any target $v \in T$ because to pebble all nodes in L_{h-1} at time t requires h parallel time steps where h pebbles are used at each time step.

Suppose for contradiction that $\mathbf{P}_{\text{opt-ss}}(\Pi_h^C, T) < 2h$. We first prove that to pebble any k targets (where $k \leq h$) simultaneously require at least k time steps (where each time step is larger than t

[18]Whereby 'simultaneously', we mean there exists some time t' where all vertices in L_{h-1} are pebbled.

defined above) where h pebbles are on the graph simultaneously. Furthermore, there exists time steps $t_{l-1} > t_{l-2} > \cdots > t_1 > t$ where h pebbles are on all vertices in L_{h-1+j} (v^i_{h-1+j} for all $i \in [1, h]$) at time t_j. We prove this by induction. Let the base case be $k = 1$. In order to pebble any target $v \in T$ using a normal strategy $\mathcal{P}$, there must be a time step $t_1 > t$ where h pebbles are on all vertices in L_h (v^i_h for all $i \in [1, h]$) by normality of pebbling strategies (see Theorem B.3 [GLT80]). We assume as our induction hypothesis that the statement is true for all $k \leq l - 1$ where $l \leq h$. We now prove the statement for $k = l$. At time t_{l-1}, there exist h pebbles on all vertices in L_{h+l-2} by definition of t_{l-1} and by our induction hypothesis. By inspection, to pebble any subset of l targets requires all vertices in L_{h+l-1} to be pebbled at some point in the execution of the pebbling strategy. Suppose there exists a strategy that pebbles k targets using at most $k - 1$ parallel moves where h pebbles are on the graph during each of the $k - 1$ parallel moves. By our induction hypothesis, pebbling any $k - 1$ sized subset of the k targets requires $k - 1$ parallel moves where h pebbles are on the graph and all nodes in L_{h+k-1} for all $k < l$ are pebbled simultaneously at time t_k. If no more than $h - 1$ pebbles can be on the vertices in L_{h+l-1}, this means that there exists a vertex in L_{h+l-1} that must be pebbled with at least l pebbles (given there exists a previous time step when h pebbles are on all vertices in L_{h+l-2} and no more than $h - 1$ of these pebbles can be moved to the vertices in L_{h+l-1}). Let this vertex be u. If we continue strategy $\mathcal{P}$ without pebbling u, then there will exists a vertex at every level $h + l' - 1$ (for all $l' \geq l$) where l' pebbles are necessary to pebble the vertex. Thus, the lower bound on the minimum number of pebbles necessary to pebble k targets using strategy $\mathcal{P}$ is $h - 1 + l'$ at some time step $t_{l'} > t_{l-1}$, a contradiction since $l' \geq 1$[19].

Given that to pebble any k targets requires at least k time steps (inaddition to the h timesteps necessary to pebble all nodes in L_{h-1}) where h pebbles are on the graph simultaneously. Thus, pebbling all targets using any strategy that pebbles sequentially subsets of targets $S_1, \dots, S_d$ where $\bigcup_{i=1}^{d} S = T$ results in $\sum_{i=1}^{d} |S_i| \geq h$ steps where h pebbles are on the graph simultaneously. In all cases, we reach a contradiction with $\mathbf{P}_{\text{opt-ss}}(\Pi^C_h, T) < 2h$. Therefore, $\mathbf{P}_{\text{opt-ss}}(\Pi^C_h, T) \geq 2h$. □

Theorem 5.7. *Using the rules of the standard pebble game, h pebbles are necessary for at least h parallel steps to pebble* any *target of a height $2h$ cylinder graph, Π^C_h.*

Proof. To pebble any target of Π^C_h requires h pebbles on all nodes in level h by normality of pebbling strategies. Given at most h pebbles, to pebble any subset k of nodes in level h (by the normality of pebbling strategies) require h pebbles to be present on the graph for at least k parallel time steps as proven in the proof for Lemma 5.6. Thus, given a pebbling strategy that pebbles the following subsets of nodes in level h sequentially, $S_1, \dots, S_d$ where $T = \bigcup_{i=1}^{d} S_i$, the number of time steps where h pebbles are on the graph is given by $\sum_{i=1}^{d} |S_i| \geq h$. Therefore, h pebbles are on the graph during at least h time steps when pebbling any target of Π^C_h, proving our theorem. □

Theorem 5.8. $\mathbf{P}_{\text{s}}(\Pi^C_h, |T|, T) \geq h$ *where Π^C_h is defined as in Def. 5.4 where $|S| = |T| = h$.*

Proof. Assume for the sake of contradiction that $s < h$ pebbles can be used to pebble all target nodes in T. By the rules of the black-magic pebble game, we can choose to use either magic pebbles or black pebbles at each time step in a valid strategy.

We first prove that given $s < |T|$ magic pebbles, one would choose to place the pebbles on s target nodes as opposed to any number of intermediate nodes. Let L_i be the set of nodes at the $i + h$-th level of Π^C_h (for $0 \leq i \leq h - 1$) where T is at level $2h - 1$ and S is at level 0. Given s adjacent pebble placements on nodes in L_i, we can pebble at most $j \leq \max(0, s + i - h + 1)$ target nodes by construction of Π^C_h without performing any repebbling of any nodes in S. (Note that we

[19]Note that a simpler proof can be shown to state that at least h pebbles are needed to pebble u at level l' but we present the present proof to show that even for a cylinder with height h (instead of $2h$) our proof here still holds–i.e. h steps where h pebbles are on the cylinder are necessary to pebble all targets T

do not need to account for the case when $s < |T|$ pebbles are placed on levels 0 to $h-1$ since no targets can be pebbled if that is the case.) If repebbling of any node in L_i needs to be done (using black pebbles), then at least h total pebbles are necessary to pebble T. We now show this is true. Suppose that in order to pebble a target node $v \in T$, there exist at most $h-i-1$ magic pebbles on adjacent nodes in L_i. Then, at least 1 additional pebble is necessary at some node in L_i to pebble v. Let the node that needs to be pebble in L_i be w. Suppose that we use a black pebble to pebble w at level i (i.e. we wouldn't choose to use magic pebbles to pebble the ancestors of w since that would use more magic pebble than if we used a magic pebble to pebble w). Note that w is the apex of a pyramid of height at least $i+1$. Therefore, at least $i+1$ black pebbles are necessary to pebble w resulting in $i+1+h-i-1=h$ total pebbles necessary to pebble v, which is greater than the initial $h-i-1$ magic pebbles in total pebble count for all $i \in [0, h-1]$ (our desired range of values of i). Note that this argument applies recursively to any number $i' \leq i$ missing pebbles at level i.

Therefore, for any number of magic pebbles $s' \leq s$ that are *not* on target nodes, we can obtain at most $s'-1$ target values without performing repebbling of any nodes in S. It is then strictly more efficient to pebble s' target nodes with magic pebbles instead of s' non-target nodes. We can have a total of $s < h$ magic pebbles which is not enough pebbles to pebble all the target nodes. To pebble the target node that is not pebbled by a magic pebble, we require h additional pebbles by pebbling price of pyramids [GLT80], contradicting our assumption. □

As a simple extension of our theorem and proof above, we get Corollary 5.9. Moreover, as an extension of the proof given for Theorem 5.8 that all magic pebbles are placed on targets and from Theorem 5.7, we obtain Corollary 5.10.

Corollary 5.9. *Given a cylinder $G = (V, E)$ as constructed in Graph Construction 5.4, G is incrementally hard:* $\mathbf{P}_{\mathrm{s}}(G, |C|-1, C) \geq |T|$ *for any subset $C \subseteq T$.*

Corollary 5.10. *Given a cylinder $G = (V, E)$ as constructed in Graph Construction 5.4,* $\mathbf{P}_{\text{opt-ss}}(G, |C|-1, C) = \Theta(|T|)$ *for all subsets of $C \subseteq T$.*

A logical question to ask after constructing our very simple hash function based on a cylinder graph is whether such a construction is optimal in terms of graph-optimal sustained complexity *and* follows our requirements for a static-memory-hard hash function. As it turns out, the graph-optimal sustained complexity of a cylinder graph is optimal in the class of layered graphs. In other words, if we choose to use layered graphs in our constructions, then we cannot hope to get a better memory and time guarantee. From an implementation and practical standpoint, layered graphs are easier to implement and hence this result has potential practical applications (as more complicated constructions need to consider memory allocation factors in the real-life implementation, not considered in the theoretical model).

Theorem 5.11. *Given a* layered graph, *$G = (V, E)$, if the number of target nodes is $|T| = s$ and* $\mathbf{P}_{\mathrm{s}}(G, s, T) \geq s$, *then $|V| = \Omega(s^2)$. A* layered graph *is one such that the vertices can be partitioned into layers and edges only go between vertices in consecutive layers.*

Proof. In order to satisfy $\mathbf{P}_{\mathrm{s}}(G, s, T) \geq s$, the number of targets has to be at least s; if $|T| < s$, then T can be completely pebbled with less than s magic pebbles and $\mathbf{P}_{\mathrm{s}}(G, s, T) < s$. Suppose the sources (the first level) are at level 0 and the targets (the last level) are at level $h-1$ where h is the height of the layered graph. In any layered graph with in-degree 2, the cost of pebbling a vertex v_i in level i is at most $i+1$ [Nor15]. Therefore, the height of G must be at least $s-1$, in order for $\mathbf{P}_{\mathrm{s}}(G, s, T) \geq s$. Let $h = s-1$. In order for $\mathbf{P}_{\mathrm{s}}(G, s, T) \geq s$, the width of the layered graph in layer j for all $j \in \left[\frac{h}{2}, h-1\right]$ must be at least $\frac{h}{2}$ (where by width, we mean the number of nodes in layer j).

Suppose that a layer j where $j \in \left[\frac{h}{2}, h-1\right]$ has width less than $\frac{h}{2}$. We can subsequently use less than $\frac{h}{2}$ magic pebbles to pebble layer j. Then, at most $\frac{h}{2}$ black pebbles are necessary to pebble all targets in T resulting in $\mathbf{P}_{\mathrm{s}}(G, s, T) < h$ and $\mathbf{P}_{\mathrm{s}}(G, s, T) < s$ (by our definition of h), a contradiction. The total number of nodes in layers $[\frac{h}{2}, h-1]$ must then be at least $\frac{h^2}{4}$, and $|V| = \Omega(h^2) = \Omega(s^2)$. $\square$

Thus, our construction of the cylinder graph is optimal in terms of amount of memory used in the asymptotic sense for the class of layered graphs. An open question is whether this is also optimal when we consider the larger class of all DAGs.

Open Question. *Does Thm 5.11 also hold for general graphs with bounded in-degree* 2*?*

Given the impossibility of providing a better space guarantee for layered graphs, we provide a general (non-layered) construction that transforms a graph from a certain class into another graph with the same space guarantee as in Theorem 5.11. Furthermore, we provide an example below that has the same space guarantees but a better time guarantee.

5.1.3 Layering *shortcut-free* graphs

We now show how to convert any *shortcut-free* DAG, $G = (V, E)$, with $\mathbf{P}_{\mathrm{s}}(G, T) = s$ and one target node (i.e. $|T| = 1$) into a DAG, $G' = (V', E')$, with $|T'| = s$ targets and $\mathbf{P}_{\mathrm{s}}(G', s, |T'|) = s$.

Definition 5.12 (Shortcut-Free Graphs)**.** *Let $G = (V, E)$ be a DAG where $\mathbf{P}_{\mathrm{s}}(G, T) \geq s$. Let $t_s^{\mathcal{P}}$ be the last time step that exactly s pebbles must be on G during any normal and regular pebbling strategy, $\mathcal{P}$, (see Thms B.3 and B.5, [GLT80, DL17]) that uses s pebbles. More specifically, let Let X be the union of the set of nodes that are pebbled at $t_s^{\mathcal{P}}$ for all normal and regular strategies $\mathcal{P}$: $X = \bigcup_{\mathcal{P} \in \mathbb{P}} P_{t_s^{\mathcal{P}}}$. Let D be the set of descendants of nodes of X. A DAG is* shortcut-free *if $|X| \leq s$ and given $s_1 < s$ pebbles placed on any subset $X_1 \subset X$, no normal and regular strategy uses less than $s - s_1$ pebbles to pebble $D \cup (X \backslash X_1)$.*

Graph Construction 5.13. *Given a shortcut-free DAG, $G = (V, E)$, with $\mathbf{P}_{\mathrm{s}}(G, T) = s$ and $|T| = 1$, we create a DAG, $G' = (V', E')$, with the following vertices and edges and with the set of targets T' where $|T'| = s$. Let X be defined as in Definition 5.12.*

1. *V' is composed of the nodes in V and $s - 1$ copies of $X \cup D$. Let the i-th copy of X be X_i (the original is X_0) and let the i-th copy of $x \in X_i$ be x_i.*
2. *E' is composed of the edges in E and the following directed edges. If $(v, w) \in E$ and $v, w \in X$, then create edges $(v_i, w_i) \in E'$ for all $i \in [1, s-1]$. Create edges $(u, v_i) \in E'$ if $(u, v) \in E$ and $u \in V \backslash X, D$.*
3. *The set of targets T' is the union of the set of targets of the different copies: $T' = \bigcup_{i=0}^{s-1} T_i$.*

Using the above construction, we have created a graph $G' = (V', E')$ where $|V'| = |V| + (s - 1)(|D| + |X|)$ and $|T'| = s$.

Theorem 5.14. *Given a shortcut-free DAG $G = (V, E)$ with $\mathbf{P}_{\mathrm{s}}(G, T) = s$ and $|T| = 1$, the construction produced by Graph Construction 5.13 produces a DAG $G' = (V', E')$ such that $\mathbf{P}_{\mathrm{s}}(G', s, |T|) = s$.*

Proof. We first prove that $\mathbf{P}_{\mathrm{s}}(G', s, |T|) \leq s$. Since there are s different targets, $\mathbf{P}_{\mathrm{s}}(G', s, |T|) \leq s$ trivially.

We now prove that $\mathbf{P}_{\mathrm{s}}(G', s, |T|) \geq s$. If only black pebbles are used to pebble the targets in T', then s black pebbles must trivially be used provided $\mathbf{P}_{\mathrm{s}}(G, T) = s$. Suppose some number of magic pebbles are used. Using the magic pebbles on any node in a copy of D (defined in Def. 5.13) that is not a target in T' is strictly worse than using a magic pebble on a target. Suppose the total number of pebbles used is less than s. We first prove that no magic pebbles are used on copies of D. If the total number of pebbles used is less than s, then not all of the s targets can be pebbled using magic pebbles. The remaining target that is not pebbled must be pebbled using s black pebbles since $\mathbf{P}_{\mathrm{s}}(G, T) = s$ by definition. By the same logic, no magic pebbles are used on the nodes in the copies of X.

Therefore, if less than s magic pebbles are used to pebble the graph, all magic pebbles should be used to pebble the predecessors of X. No magic pebble can be removed and repebbled since such a magic pebble must be placed s times (once for each copy of X and D), exceeding the maximum number of magic pebbles we can have. Given that we can use a total of less than s magic pebbles to pebble the predecessors of X, suppose some $s' < s$ pebbles are used, then less than $s - s'$ pebbles are left to pebble each copy of X and D; by incremental hardness, less than $s - s'$ cannot be used to pebble each copy of X and D. At least one magic pebble is used on the predecessors of X; by our definition of incremental hardness, less than $s - 1$ pebbles cannot be used to pebble X and D, a contradiction. Thus, $\mathbf{P}_{\mathrm{s}}(G', s, T) \geq s$. □

If $D = \Theta(s)$ and $s = O(\sqrt{|V|})$, then $|V'| = \Theta(s^2 + |V|)$ which has a better sustained time guarantee than our cylinder construction.

We first note that the sustained memory graphs presented in [ABP17a] *do not* achieve optimal local memory hardness because $X \cup D$ (as defined in Definition 5.13) is $\Theta(n)$ (since the sources are the ones that remain pebbled in their construction). Thus, we would like to provide a construction of a shortcut-free DAG where $|X \cup D| = \Theta(s)$. Note that the size of $X \cup D$ will always be $\Omega(s)$, trivially. We now provide a definition of a shortcut-free graph class G that can be transformed using Definition 5.13.

Graph Construction 5.15 (Illustrated in Fig. 3)**.** *Let $G = (V, E)$ be a graph defined by parameter s and in-degree 2 with the following set of vertices and edges:*

1. *Create a height s pyramid. Let r_i be the root of a* subpyramid *(i.e. a pyramid that lies in the original height s pyramid) with height $i \in [2, s]$. One can pick any set of these subpyramids.*
2. *Topologically sort the vertices in each level and create a path through the vertices in each level (see Fig. 3). Replace any in-degree-3 nodes with a pyramid of height 3, with a 6-factor increase in the number of vertices.*
3. *Create $c_1 s$ additional nodes for some constant $c_1 \geq 2$ (in Fig. 3, $c_1 = 6$). Label these nodes v_j for all $j \in [1, c_1 s]$.*
4. *Create directed edges (r_s, v_1) and $(r_i, v_{k(i-1)})$ for all $k \in [1, s]$.*
5. *Create $s - 1$ additional nodes. Let these nodes be w_l for all $l \in [1, s-1]$.*
6. *Create directed edges $(v_{c_1 s}, w_1)$ and (r_i, w_{i-1}) for all $i \in [2, s]$.*
7. *The target node is w_{s-1}.*

Lemma 5.16. *Given a DAG $G = (V, E)$ and a parameter s where G is defined by Definition 5.15, $\mathbf{P}_{\mathrm{s}}(G, T) = s$.*

Proof. In order to pebble the apex of the pyramid of height s, we must use at least s pebbles as proven in the proof for black pebbling cost of pyramids [GLT80]. □

Before we prove that $G = (V, E)$ created by Definition 5.15 with parameter s is shortcut-free, we first prove the following stronger lemma which will help us prove that G is shortcut-free.

Lemma 5.17. *Let $G = (V, E)$ be a graph created using Definition 5.15 with parameter s. Given a normal strategy $\mathcal{P}$ to pebble G, when v_q for $q \in [1, c_1 s]$ is pebbled at some time step, black pebbles are present on all nodes in $[r_i, r_s]$ where $i = (q \bmod s - 1) + 1$ from the time when v_1 is pebbled to when v_q is pebbled.*

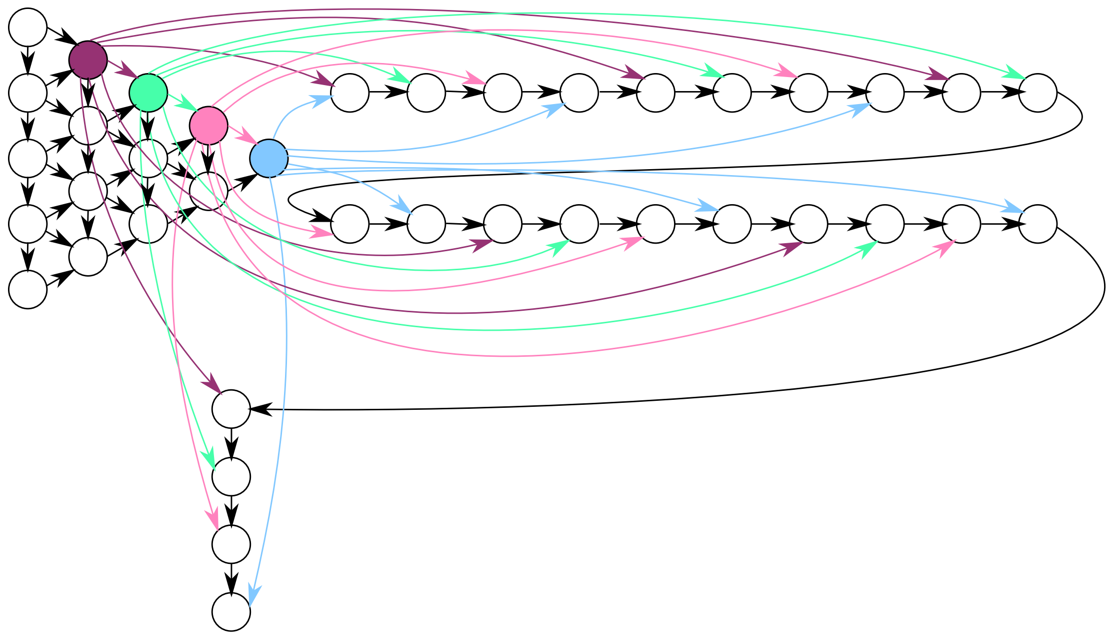


Figure 3: Example of a time optimal graph family construction as defined in Def. 5.15. Here, $s = 5$.

Proof. We prove this lemma via induction.

In our base case when $i = s$, when the corresponding v_q is pebbled, a black pebble must be on r_s and v_{q-1} in the previous time step. Thus, a black pebble remains on r_s from the time v_1 is pebbled till the time that v_q is pebbled or a set of $s - j$ black pebbles remain on the j-th level of the pyramid for some $j \in [0, s-1]$ (in which case we can charge one of these pebbles to be "present on r_s"). Suppose neither of these conditions are met. Then, by the pebbling number of pyramids (see Thm B.6, [Nor15]), at least s pebbles must be used to pebble r_s, contradicting the frugality of $\mathcal{P}$ (since at most s pebbles are used to pebble G). In general, we make the observation that if there $s - j$ pebbles on some level $j \in [0, s-1]$, then we can charge these $s - j$ pebbles to be "on all nodes in $[r_j, r_s]$".

For our induction hypothesis, we assume that the theorem is true for j and prove the stratement for $i = j - 1$. When $i = j - 1$ and the corresponding v_q is pebbled, we assume by our induction hypothesis that there are $s - j$ black pebbles present on $[r_j, r_s]$ (or charged to be on $[r_j, r_s]$) from when v_1 is pebbled to when v_q is pebbled. In order to pebble v_q, there must be black pebbles on r_i and v_{q-1}. If there does not exist a black pebble on r_i (or on the predecessors of r_i) from when v_1 is pebbled to when v_q is pebbled, then at least one pebble must be removed from some $r \in [r_j, r_s]$ or from v_{q-1} since at least $j - 1$ pebbles are necessary to pebble r_{j-1} ($s - j + 2$ pebbles are currently in use–leaving not enough pebbles to pebble r_{j-1} unless a pebble is removed). If the black pebble is removed from v_{q-1}, the frugality of $\mathcal{P}$ is contradicted. If the black pebble is removed from some $r \in [r_j, r_s]$, then by observation, r_s will need to be repebbled sometime in the future, also a contradiction to the frugality of $\mathcal{P}$. Thus, we prove our statement. □

Lemma 5.18. *Given a DAG $G = (V, E)$ and a parameter s where G is defined by Definition 5.15, G is shortcut-free.*

Proof. We first prove that any normal standard pebbling strategy $\mathcal{P}$ that pebbles G must contain pebbles on all r_i and $v_{c_1 s}$ at some time (say, t_X) during the execution of $\mathcal{P}$.

Let X be the set of vertices containing black pebbles when $v_{c_1 s}$ is pebbled. Thus, a total of s pebbles must be on the graph (specifically on all nodes in X) at this time in any normal strategy by proof of Lemma 5.17. We now prove the incremental hardness of G. Let $s' < s$ pebbles be on X at time t_X. We prove that we cannot pebble $X \cup D$ using less than $s - s'$ pebbles.

Suppose for the purposes of contradiction, given $s' < s$, assume that s' pebbles are placed on X and less than $s - s'$ pebbles can be used to pebble $X \backslash X' \cup D$. Supose that $X \backslash X'$ includes either:

1. $v_{c_1 s}$ and some $s - s' - 1$ subset of vertices in $[r_2, r_s]$, or
2. some $s - s'$ subset of vertices in $[r_2, r_s]$.

In the first case, if no pebbles are on v_i for $i \in [1, c_1 s]$, then at least one pebble needs to be used to pebble v_i for $i \in [1, c_1 s]$. If $X \backslash X'$ includes some $s - s' - 1$ subset of vertices in $[r_2, r_s]$, then at least $s - s'$ pebbles are needed to pebble the vertices missing the pebbles.

In the second case, if some subset $s - s'$ of vertices in $[r_2, r_s]$ are in $X \backslash X'$, then at least $s - s' + 1$ pebbles are necessary to pebble the nodes missing pebbles in order to be able to pebble w_l for $l \in [1, s-1]$.

In either case, at least $s - s'$ pebbles are necessary to pebble $X \backslash X' \cup D$, thus, this construction is shortcut-free. □

Theorem 5.19. *s pebbles are necessary for at least $\Theta(s^2)$ parallel steps to pebble any target of G'.*

Proof. To pebble v_j for all $j \in [1, c_1 s]$, we require pebbles on all r_i for $i \in [2, s]$ and one pebble on the path from v_1 to $v_{c_1 s}$; otherwise, the entire pyramid must be rebuilt, resulting in repebbling all nodes in the graph as we showed in the proof of Lemma 5.17. To pebble the pyramid requires s pebbles on the pyramid at all times and takes $\Theta(s^2)$. We show this is true.

Suppose that at some point before pebbling the apex of the pyramid that a pebble is removed from the graph, then, by our requirement that $s - 1$ pebbles must remain on r_i for $i \in [2, s]$ and that a pebble must be on the path from v_1 to $v_{c_1 s}$, the removed pebble cannot be used for either of these tasks. Thus, the entire pyramid must be rebuilt, contradicting the frugality of the strategy.

Thus, s nodes must remain on the graph for $\Theta(s^2 + c_1 s) = \Theta(s^2)$ parallel time steps, proving our theorem. □

We create $G' = (V', E')$ from G (as constructed using Definition 5.15) using Definition 5.13 , resulting in a graph with $\Theta(s^2)$ total nodes.

Theorem 5.20. $\mathbf{P}_{\mathrm{s}}(G', s, T) = s$.

Proof. By Lemma 5.18 the graph is shortcut-free and by Lemma 5.16 $\mathbf{P}_{\mathrm{s}}(G, T) = s$, therefore, we use Theorem 5.14 to prove that $\mathbf{P}_{\mathrm{s}}(G', s, T) = s$. □

By the proof that G' is shortcut-free, we obtain the following corollary that G' is also incrementally hard. Moreover, Corollary 5.22 follows directly from the proof of Theorem 5.14.

Corollary 5.21. *Given a graph $G = (V, E)$ as constructed in Graph Construction 5.15, G is incrementally hard: $\mathbf{P}_{\mathrm{s}}(G, |C| - 1, C) \geq |T|$ for any subset $C \subseteq T$.*

The following corollary about the graph-optimal sustained time complexity is proven directly from the proof of Lemma 5.17 and Theorem 5.19 that if less than $\frac{s}{2}$ magic pebbles are on the pyramid, then half the pyramid must be rebuilt resulting in $\Theta(s^2)$ time-steps in which s pebbles are on the graph; thus proving for the cases when $|C| - 1 < \frac{s}{2}$. We now prove the case when $|C| - 1 \geq \frac{s}{2}$.

Corollary 5.22. *Given a graph $G = (V, E)$ as constructed in Graph Construction 5.15, $\mathbf{P}_{\text{opt-ss}}(G, |C| - 1, C) = \Theta(|V|)$ for all subsets of $C \subseteq T$.*

Proof. If $|C| - 1 \geq \frac{s}{2}$ magic pebbles are not placed on r_i for all $i \in [2, s]$, then we have to rebuild at least half the pyramid, resulting in $\Theta(s^2) = \Theta(|V|)$ time being used. Thus, some $s' \geq \frac{s}{2}$ magic pebbles must be used on r_i for all $i \in [2, s]$. Then, to pebble all $|C| \geq \frac{s}{2}$ targets requires $\Theta(s^2) = \Theta(|V|)$ time using another black pebble since $s' \geq \frac{s}{2}$ pebbles are used on the pyramid. □

5.2 $\mathcal{H}_2$ construction

Our construction of $\mathcal{H}_2$ is presented in Algorithm 1.

Algorithm 1 $\mathcal{H}_2$

```
On input $(1^\kappa, x)$ and given oracle access to $\mathsf{Seek}_R$ (where $R$ is the string outputted by $\mathcal{H}_1$):
 1. Let $[\![R]\!] = |R|/w$ be the length of $R$ in words.
 2. Query the random oracle to obtain $\rho_0 = \mathcal{O}(x)$ and $\rho_1 = \mathcal{O}(x+1)$.
 3. Use $\rho_0$ to sample a random $\iota \in [[\![R]\!]]$.
 4. Query the $\mathsf{Seek}_R$ oracle to obtain $y' = \mathsf{Seek}_R(\iota)$.
 5. Output $y' \oplus \rho_1$.
```

Lemma 5.23. *For any R, the output distribution of $\mathcal{H}_2$ is uniform over the choice of random oracle $\mathcal{O} \leftarrow \mathbb{O}$.*

Proof. Over the choice of random oracle, the value ρ_1 computed in Step 2 is truly random, and y' is independent of ρ_1 by construction, so the output $y' \oplus \rho_1$ is also truly random. □

Remark. Lemma 5.23 is important as an indication that our SHF construction "behaves like a random oracle." The memory-hardness guarantee alone does not assure that the hash function is suitable for cryptographic hashing: e.g., a modified version of $\mathcal{H}_2$ which directly outputted y' instead of $y' \oplus \rho_1$ would still satisfy memory-hardness, but would be an awful hash function (with polynomial size codomain). The inadequacy of existing memory-hardness definitions for assuring that a function "behaves like a hash function" is discussed by [AT17].

5.3 Proofs of hardness of SHF Constructions

We now prove the hardness of our graph constructions given earlier in Section 5.

We begin by stating two supporting lemmata. The first is due to Erdős and Rényi [ER61], on the topic of the Coupon Collector's Problem.

Lemma 5.24 ([ER61])**.** *Let Z_n be a random variable denoting the number of samples required, when drawing uniformly from a set of n distinct objects with replacement, to draw each object at least once. Then for any c, $\lim_{n\to\infty} \Pr[Z_n < n \log n + cn] = e^{-e^{-c}}$.*

Corollary 5.25. *Let $Z_{n,k}$ be a random variable denoting the number of samples required, when drawing uniformly from a set of n distinct objects with replacement, to have drawn at least $k \in [n]$ distinct objects. Let $q \in \omega(k \log k)$. Then $\Pr[Z_{n,k} < q]$ is overwhelming (in k).*

Proof. For $m \in \mathbb{N}$ and $i \in [m-1]$, let $\mathcal{E}_{i,m}$ denote the event that after i elements out of a set of m elements have already been sampled uniformly with replacement, the $(i+1)$th sample will coincide with one of the elements already drawn. For any $i \le k \le n$, it holds that $\Pr[\mathcal{E}_{i,n}] \ge \Pr[\mathcal{E}_{i,k}]$. The desired event of drawing k distinct objects corresponds exactly to the conjunction of $\mathcal{E}_{i,m}$ for $i \in [k]$. Therefore, for all $k \in [n]$ and any c',

$$\Pr[Z_{n,k} < c'] \ge \Pr[Z_k < c'] \ . \tag{4}$$

Hence, it suffices for our purposes to bound $\Pr[Z_k]$. From Lemma 5.24,

$$\lim_{k\to\infty} \Pr[Z_k < k \log k + ck] = \lim_{k\to\infty} e^{-e^{-c}}.$$

Applying a Taylor expansion, we get $\Pr[Z_k < k \log k + ck] \in O(1 - e^{-c})$. This probability is overwhelming in k (i.e., e^{-c} is negligible) whenever $c \in \omega(\log(\kappa))$. □

Theorems 5.26–5.29 state the static-memory-hardness of our SHF constructions based on Graph Constructions 5.4 and 5.15.

Theorem 5.26. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}_{\Pi_h^C}$ (Graph Construction 5.4), and let $\mathcal{H}_2$ be as defined in Algorithm 1. Let $\mathcal{H} = \{h_\kappa\}_{\kappa\in\mathbb{N}}$ be the static-memory hash function family described by $(\mathcal{H}_1, \mathcal{H}_2)$. Let $\hat{\kappa} = \kappa - \xi\log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda}, \tau \in \Theta(\sqrt{n})$, and let $q \in \omega(\Lambda\log\Lambda)$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

Proof. Suppose, for contradiction, that the theorem does not hold. Then by Definition 4.2, there exist: $\kappa \in \mathbb{N}$, a string $R \in \{0,1\}^{\hat{\kappa}(\hat{\Lambda}-1)}$, an algorithm $\mathcal{A}$, and a set $X = \{x_1, \ldots, x_q\}$ such that the following probability is non-negligible:

$$\Pr_{\mathcal{O},\rho}\left[\left\{(x_1, h_\kappa(x_1)), \ldots, (x_q, h_\kappa(x_q))\right\} = \mathcal{A}(1^\kappa, R; \rho) \wedge \textsf{s-mem}_{\mathbb{O}}(\hat{\kappa}\hat{\Lambda}, \mathcal{A}, R, \rho) < \tau\right] . \tag{5}$$

We denote by $\mathcal{E}_\rho$ the event that

$$(x_1, h_\kappa(x_1)), \ldots, (x_q, h_\kappa(x_q))\} = \mathcal{A}(1^\kappa, R; \rho) \wedge \textsf{s-mem}_{\mathbb{O}}(\hat{\kappa}\hat{\Lambda}, \mathcal{A}, R, \rho) < \tau .$$

Given a correct evaluation $y = h_\kappa(x)$ of $\mathcal{H}_2$ on a given input x, one can easily compute ρ_0, ρ_1 by evaluating $\mathcal{O}$ on $x, x+1$ respectively, and demask y to obtain the value $y'(x) = y \oplus \rho_1$ of the target label computed in Step 4 of Algorithm 1. Moreover, the index ι computed in Step 3 can be computed as a deterministic function of ρ_0. Define $\mathcal{B}'$ to be the deterministic algorithm that on input (x, y) computes ρ_0, ρ_1 and y' as described above, and outputs (ι, y').

Next, define $\mathcal{B}$ to be the algorithm that runs $\mathcal{A}$ and then applies $\mathcal{B}'$ on each pair (x_i, y_i) outputted by $\mathcal{A}$, and outputs the resulting set $J = \{(\iota_1, y_1'), \ldots, (\iota_q, y_q')\}$ where each $(\iota_i, y_i') = \mathcal{B}'(x_i, y_i)$. By construction, if $y_i = h_\kappa(x_i)$, each y_i' is the correct label of ι_ith target node of the cylinder graph. Notice that this means that for each value of ι, there is a unique value of y' such that $(\iota, y') = \mathcal{B}'(x, h_\kappa(x))$ for any x.

Let I denote $|\{\iota_i\}_{x_i\in X}|$. Since the set X is fixed before the random oracle, the locations I are distributed uniformly and independently (with replacement). Then by Corollary 5.25, the number of distinct locations $|I|$ is at least $\hat{\Lambda}$ with overwhelming probability. That is, there is a negligible function ε' such that $\Pr\left[|I| \geq \hat{\Lambda}\right] \geq 1 - \varepsilon'$. Conditioned on $\mathcal{E}_\rho$, all pairs (x_i, y_i) outputted by $\mathcal{A}$ are such that $y_i = h_\kappa(x_i)$, and we have already observed that each value of ι induces a unique value of y' outputted by $\mathcal{B}'$ on input pairs of the form $(x_i, h_\kappa(x_i))$. It follows that $\Pr[|J| \geq \hat{\Lambda} \mid \mathcal{E}_\rho] \geq 1 - \varepsilon'$.

Now consider the ex-post-facto magic pebbling strategy $\mathcal{P}$ induced by $\mathcal{B}$. By Lemma 3.8, with overwhelming probability over the random oracle and the coins of $\mathcal{B}$, $\mathcal{P}$ is legal and uses at most

$$\left\lfloor\frac{|R|}{\hat{\kappa}}\right\rfloor = \left\lfloor\frac{\hat{\kappa}(\hat{\Lambda}-1)}{\hat{\kappa}}\right\rfloor \leq \hat{\Lambda} - 1 \tag{6}$$

magic pebbles; call this event $\mathcal{E}_\rho'$ (where ρ denotes the randomness of $\mathcal{B}$). By Lemma 3.9, with overwhelming probability over the same,

$$\forall i \in [t],\ |P_i| \leq \left\lfloor\frac{|\sigma_i|}{\hat{\kappa}}\right\rfloor , \tag{7}$$

where t is the length of $\mathcal{P}$, P_i is the ith configuration of $\mathcal{P}$, and σ_i is the ith state of the execution of $\mathcal{B}$. We denote by $\mathcal{E}_\rho''$ the event that (7) is satisfied (where ρ denotes the randomness of $\mathcal{B}$). By definition, event $\mathcal{E}_\rho$ implies that $|\sigma_i| \geq \hat{\kappa}\hat{\Lambda}$ for fewer than τ values of i. Combining this observation with (7), we have that whenever $\mathcal{E}_\rho$ occurs, $|P_i| \geq \hat{\Lambda}$ for fewer than τ values of i.

Finally, we observe that conditioned on $\mathcal{E}_\rho$, since we established above that $\mathcal{B}$ outputs a set of at least $\hat{\Lambda}$ correct target labels, the strategy $\mathcal{P}$ must successfully pebble the corresponding $\hat{\Lambda}$ target nodes. Since $\Pr[\mathcal{E}_\rho]$ is non-negligible and $\Pr[\mathcal{E}']$ and $\Pr[\mathcal{E}'']$ are overwhelming, $\Pr[\mathcal{E}' \wedge \mathcal{E}''|\mathcal{E}]$ must be negligibly close to $\Pr[\mathcal{E}]$ (and thus, non-negligible). The occurrence of $\mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{E}''$ implies the existence of a pebbling strategy $\mathcal{P}$ that is legal, uses at most $\hat{\Lambda} - 1$ magic pebbles, and for which the number of time-steps in which at least $\hat{\Lambda}$ total (i.e., black and magic) pebbles are used is less than τ. This contradicts Corollaries 5.9–5.10. $\square$

Theorem 5.27. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}_G$ (Graph Construction 5.15), and let $\mathcal{H}_2$ be as defined in Algorithm 1. Let $\hat{\kappa} = \kappa - \xi \log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda} \in \Theta(\sqrt{n})$, let $\tau \in \Theta(n)$, and let $q \in \omega(\Lambda \log \Lambda)$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

Proof sketch. Identical proof structure to the proof of Theorem 5.26, except instead of invoking Corollaries 5.9–5.10 at the end, we derive a contradiction to Corollaries 5.21–5.22. $\square$

The parameter q is suboptimal in Theorems 5.26 and 5.27. We can achieve optimality (i.e., $q = [\!]|R|[\!]$) by the following alternative construction of $\mathcal{H}_2$: make $q' = \omega(\log(\kappa))$ random calls instead of just one call to the Seek oracle in Step 4. To preserve the output size of h_κ, it may be useful to reduce the size of node labels by a corresponding factor of q'. This can be achieved by truncating the random oracle outputs used to compute labels in Definition A.1. The description of this altered $\mathcal{H}_2^{q'}$ and the definition of graph function family $\mathcal{F}q'_G$ with shorter labels are given in Appendix A.

Theorem 5.28. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}^{\kappa/q'}_{\Pi^C_h}$ (Graph Construction 5.4), and let $\mathcal{H}_2$ be $\mathcal{H}_2^{q'}$ as defined in Algorithm 2 for some $q' \in \omega(\log \Lambda)$. Let $\hat{\kappa} = \kappa - \xi \log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda}, \tau \in \Theta(\sqrt{n})$, and let $q = \Lambda$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

Proof sketch. Identical proof structure to the proof of Theorem 5.26, except that when invoking Corollary 5.25, due to the design of $\mathcal{H}_2^{q'}$ which calls Seek more times than $\mathcal{H}_2$, we obtain the stronger statement that an adversary that successfully outputs q pairs $((x_1, h_\kappa(x_1)), \ldots, (x_q, h_\kappa(x_q)))$ must correctly guess q target labels of the graph. $\square$

Theorem 5.29. *Define a static-memory hash function family $(\mathcal{H}_1, \mathcal{H}_2)$ as follows: let $\mathcal{H}_1$ be the graph function family $\mathcal{F}^{\kappa/q'}_G$ (Graph Construction 5.15), and let $\mathcal{H}_2$ be $\mathcal{H}_2^{q'}$ as defined in Algorithm 2 for some $q' \in \omega(\log \Lambda)$. Let $\hat{\kappa} = \kappa - \xi \log(\kappa)$ for any $\xi \in \omega(1)$, let $\hat{\Lambda} \in \Theta(\sqrt{n})$, let $\tau \in \Theta(n)$, and let $q = \Lambda$. Then $(\mathcal{H}_1, \mathcal{H}_2)$ is $(\hat{\kappa}\hat{\Lambda}, \hat{\kappa}, \tau, q)$-hard.*

Proof sketch. Identical proof structure to the proof of Theorem 5.28, except instead of invoking Corollaries 5.9–5.10 at the end, we derive a contradiction to Corollaries 5.21–5.22. $\square$

6 Capturing nonlinear space-time tradeoffs with CC^α

Next, we motivate our notion of CC^α (Definition 2.16). We show that both the honest party and the adversary may choose to use different pebbling strategies given different values of α even when α is constant. Furthermore, we show that both of our pebbling constructions of $\mathcal{H}_1$ (given in Section 5) have the desirable feature that the honest party *and* the adversary use the same strategy regardless of the size of α.

6.1 CC and CC$^\alpha$ consider cumulative cost of *different strategies*

We present a graph family with in-degree-2 where the strategy that an adversary chooses to pebble an instance G in the graph family differs depending on the α parameter of the CC$^\alpha$ complexity measure. We show that in our case, for certain α, we would choose to use constant space, whereas for other α, using superconstant space is the preferred option. We define our graph family as follows:

Graph Construction 6.1. *We define a graph family $\mathbb{G}$ with bounded degree 2 and arbitrary $n \in \mathbb{N}$ nodes such that the time-space tradeoff of a graph with n nodes in the family is $T(S) \geq (\frac{n^c}{n^a})(n^a - (S-2))(n^b) + n$ (where S is the number of pebbles used to pebble the graph) where $0 \leq a, b, c < 1$, $b + c > a + 1$, $a < b, c$, and $n^c \approx n - n^{a+b}$.*

- *Given a graph $G = (V, E)$ with n vertices, partition the set of vertices, V, into 2 sets, A and B where $|A| = n^{a+b}$ and $|B| = n^c$ (since we know $n^c \approx n - n^{a+b}$, $n^c + n^{a+b} \approx n$).*
- *We arbitrarily order all vertices in B in some order, $[v_i, \dots, v_n]$ and create edges $(v_j, v_{j+1}) \in E$ for all $j \in [i, n-1]$.*
- *We arbitrarily order all vertices in A in some order, $[v_1, \dots, v_{i-1}]$ and create edges $(v_j, v_{j+1}) \in E$ for all $j \in [1, i-2]$.*
- *We create edge (v_{i-1}, v_i).*
- *Create edges $(v_k, v_l) \in E$ ($v_k \in A$ and $v_l \in B$) where $k \bmod n^b = 0$ and $l = n^{a+b} + \left(\frac{k}{n^b}\right) + (q-1)n^a$ for all integers $q \in \left[1, \frac{n^c}{n^a}\right]$.*

Fig. 4 illustrates Graph Construction 6.1.

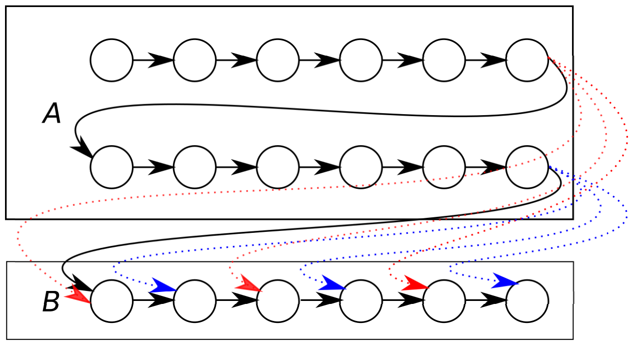


Figure 4: Graph Construction 6.1 with $n = 16$, $a = \frac{1}{4}$, $b = \frac{2}{3}$, $c = \frac{2}{3}$. For clarity, we depict $n^a = 2$, $n^b \approx 6$ and $n^c \approx 6$.

We show that there are at least two pebbling strategies, $\mathcal{P}_1$ and $\mathcal{P}_2$, where an adversary would differ in his preferred strategy depending on α when using the CC$^\alpha$ complexity measure when $\alpha > \alpha'$ where α' is calculated with respect to the parameters of the graph family constructed from Graph Construction 6.1.

Lemma 6.2. *Given a pebbling strategy $\mathcal{P}_1$ that uses constant space S_1, $\mathsf{Time}(\mathcal{P}_1) = \Theta(n^{b+c})$ where $G \in \mathbb{G}$ in defined by Graph Construction 6.1.*

Proof. Suppose that a constant S_1 pebbles can be on the graph at any particular time, then at most $S_1 - 1$ of the vertices in $A \subseteq V$ can be pebbled. It does not help to pebble the vertices in B since all vertices in B needs to be pebbled only once regardless of the pebbling strategy used. Since only the vertices $v_j \in A$ where $j \mod n^b = n^b - 1$ are connected to vertices in B, using the given S_1, the optimal placements are on vertices v_j in order to minimize pebbling time since any extra space needs to be used to pebble B and pebbling anywhere else results in greater pebbling time since the pebble needs to be moved to vertex v_j by the pigeonhole principle. Given constant S_1 pebbles, there exist v_j vertices that do not contain pebbles. Thus, each time one reaches a vertex in B with predecessor $v_j \in A$ without a pebble, at least n^b time must be spent to pebble it. Therefore, given S_1 pebbles, the total amount of time necessary to pebble G is $(\frac{n^c}{n^a})(n^a - S_1)(n^b) + n = \Theta(n^{b+c})$. □

Corollary 6.3. *Given a pebbling strategy, $\mathcal{P}_1$, that uses constant space S_1, $\mathsf{p\text{-}cc}_\alpha(\mathcal{P}_1) = \Theta(n^{b+c})$ where $G \in \mathbb{G}$ is constructed by Def. 6.1.*

Proof. This follows immediately from Lemma 6.2 since constant space is used throughout the pebbling. □

Lemma 6.4. *Given a pebbling strategy $\mathcal{P}_2$ that uses space $S_2 = n^a + 1$, $\mathsf{Time}(\mathcal{P}_2) = \Theta(n)$ where $G \in \mathbb{G}$ is constructed by Graph Construction 6.1.*

Proof. It is trivial to show that pebbling a line takes $\Omega(n)$ time since all nodes have to be pebbled at least once. We now show a strategy using $n^a + 1$ pebbles that uses $O(n)$ time.

We start with the vertices in A and pebble them in topological order, keeping pebbles on all $v_j \in A$ where $j \bmod n^b = n^b - 1$. There exists exactly n^a vertices in A by definition that are predecessors of vertices in B. Therefore, as we pebble the vertices in A in topological order, we leave a pebble on each vertex v_j. When we pebble B all predecessors of vertices in B are either in B or are pebbled in A. Therefore, we only need to pebble all vertices in A and B once, resulting in $\mathsf{Time}(\mathcal{P}_2) = \Theta(n)$. □

The following corollary is directly proven by the proof of Lemma 6.2.

Corollary 6.5. *Given a pebbling strategy, $\mathcal{P}_2$, that uses space $S_2 = n^a + 1$ and $\mathsf{Time}(\mathcal{P}_2) = \Theta(n)$, $\mathsf{p\text{-}cc}_\alpha(\mathcal{P}_2) = \Theta(n^{\alpha a+1})$ where $G \in \mathbb{G}$ is constructed by Graph Construction 6.1.*

Lemma 6.6. *When $\alpha = 1$, then $CC^\alpha(G) = \Theta(n^{a+1})$.*

Proof. Suppose in the case when $\alpha = 1$, we use a pebbling strategy, $\mathcal{P}$, that uses nonconstant space $s = o(n^a)$. Then, for each pebble, we pebble one of the vertices $v_j \in A$ where $j \bmod n^b = n^b - 1$. The resulting $\mathsf{p\text{-}cc}_\alpha(\mathcal{P}) = s(\frac{n^c}{n^a})(n^a - s)(n^b) + n$ which is minimized when $s = n^a + 1$ given $b + c > a + 1$ by definition of our graph family. □

Lemma 6.7. *For all a, b, c, there exists an α' such that for all constant $\alpha > \alpha'$, $CC^\alpha(G) = \Theta(n^{b+c})$.*

Proof. Given a pebbling strategy, $\mathcal{P}$, that uses space $s = \omega(1)$, the pebbling cost is then $\mathsf{p\text{-}cc}_\alpha(\mathcal{P}) = s^\alpha(\frac{n^c}{n^a})(n^a - s)(n^b) + n$. When $\alpha > 1$, $\mathsf{p\text{-}cc}_\alpha(\mathcal{P}) = \Theta(\min(s^\alpha n^{b+c}, n^{\alpha a+1})) = \omega(n^{b+c})$ when $\alpha > \frac{b+c}{a}$ and $s = \omega(1)$. Therefore, only for $s = O(1)$, does the pebbling cost become $\mathsf{p\text{-}cc}_\alpha(\mathcal{P}) = \Theta(n^{b+c})$ when $\alpha' = \frac{b+c}{a}$. Since a, b, c are constants, for all $\alpha > \alpha'$, $\mathrm{CC}^\alpha(G) = \Theta\left(n^{b+c}\right)$. □

From the above two lemmas, we immediately get the following theorem regarding the CC^α of the constructions given different constant values of α.

Theorem 6.8. *Given a graph $G = (V, E)$ as constructed by Graph Construction 6.1, when $\alpha = 1$, $CC^\alpha(G) = \Theta(n^{a+1})$ but when $\alpha > \alpha'$ for some constant α', $CC^\alpha(G) = \Theta(n^{b+c})$.*

As an immediate result of the above, there exists a point for constants a, b, c that the adversary chooses a different strategy to pebble a graph for different constant values of α (we can pick values of a, b, c such that α' can be reduced even down to $\alpha' \geq 3$).

6.2 Upper bounds for CC^α

We prove a tighter upper bound for CC^α when α is a constant than the trivial upper bound of $n^{\alpha+1}$. We first note that $n^{\alpha+1}$ is a trivial upper bound on the $\mathrm{CC}^\alpha(G)$ of a graph, G, since at any timestep $\mathbf{P}_{\mathrm{s}}(G,T) \leq n$ and the algorithm runs for $\mathsf{Time}(G,|T|) \leq n$ given n space is used throughout. Therefore, $\mathrm{CC}^\alpha(G) \leq n^{\alpha+1}$ for all graphs G. We now prove a tighter upper bound using the general pebbling algorithm described in [AB16] as $\mathsf{GenPeb}(G,S,g,d)$.

We formulate a simplified version of the $\mathsf{GenPeb}(G,S,g,d)$ procedure which we call the $\mathsf{GenPeb}(G)$ procedure. At a high-level the $\mathsf{GenPeb}(G)$ algorithm proceeds as follows (see [AB16] for more detail).

Definition 6.9 ($\mathsf{GenPeb}(G)$:)**.**

1. *There exists a subset S of $|S| \leq \frac{2\alpha n \log\log n}{\log n}$ vertices (for large enough n where $2\alpha \log\log n \leq \log n$) such that $\mathsf{depth}(G-S) \leq \frac{n}{\log^\alpha n}$ (Lemma 6.1, 6.2 in [AB16], [Val77]).*
2. ***Balloon Phase:** Pebble all nodes up to depth $\frac{n}{\log^\alpha n}$ (depth measured from the last light phase) until all immediate descendants lie in S.*
3. ***Light Phase:** When all immediate descendants lie in S, remove all pebbles from nodes not in S and not on parents of the next nodes to be pebbled. Continue in the light phase until a node not in S must be pebbled.*
4. *Repeat the above until no more nodes need to be pebbled.*

Lemma 6.10. *Let s_Σ be the total number of pebbles used in the balloon phase (the sum of the number of pebbles used in all balloon phases) and $s_{\Sigma\backslash S}$ be the total number of pebbles used in the balloon phases on all nodes $v \notin S$. Then, $s_{\Sigma\backslash S} \leq n$.*

Proof. This proof is trivial since at most n pebbles can be the graph at any time. □

Lemma 6.11. *Let $\Sigma\backslash S$ be the subgraph of $G=(V,E)$ which is pebbled during the balloon phase and whose vertices are not in S. Then, $CC^\alpha(\Sigma\backslash S) \leq \frac{n^{\alpha+1}}{\log^\alpha n}$.*

Proof. By Lemma 6.10, the number of pebbles necessary to pebble $\Sigma\backslash S$ is at most n: $s_{\Sigma\backslash S} \leq n$. Therefore, we can compute $\mathrm{CC}^\alpha(\Sigma\backslash S) \leq \sum_{B_i\in\mathcal{B}}(|B_i|)^\alpha \leq n^\alpha(\frac{n}{\log^\alpha n}) = \frac{n^{\alpha+1}}{\log^\alpha n}$ given a series of balloon phase pebble configurations $\mathcal{B}$ where $\sum_{B_i\in\mathcal{B}}|B_i| = n$ and $B_0 \biguplus \cdots \biguplus B_{|\mathcal{B}|} = V$. □

Lemma 6.12. *Let $CC^\alpha(S)$ be the cost of pebbling S in both the light and the balloon phases. The $CC^\alpha(S)$ of the light and balloon phases is at most $O\left(\frac{n^{\alpha+1}(\log\log n)^\alpha}{\log^\alpha n}\right)$.*

Proof. The total amount of time that light and balloon phases last in which nodes in S are pebbled is at most n timesteps since a number greater than n implies that $|S| \geq n$ which is impossible since the number of nodes in the graph is n. In the light phases, at most $2|S| = \frac{4\alpha n \log\log n}{\log n}$ pebbles are kept on the graph since each node has bounded in-degree 2. Therefore, $\mathrm{CC}^\alpha(G) \leq \frac{4^\alpha \alpha^\alpha n^{\alpha+1}(\log\log n)^\alpha}{\log^\alpha n}$. □

Theorem 6.13. *For any bounded in-degree-2 graph, $CC^\alpha(G) = O\left(\frac{n^{\alpha+1}(\log\log n)^\alpha}{\log^\alpha n}\right)$ for constant $\alpha \geq 1$.*

Proof. This follows directly from Lemmas 6.11 and 6.12. □

6.3 Asymptotically tight sequential lower bound for $\alpha = 1$

We give an explicit construction of a graph that achieves asymptotically tight lower bound (up to $\log\log n$ factors) in CC^α that matches our upper bound provided in Section 6 for $\alpha = 1$ and in [AB16, ABP17b] *when considering the sequential pebbling model*[20] . Previous constructions [AB16, ABP17a] ignored $\log\log n$ factors and were not tight up to such factors in the parallel model. Because we consider the sequential pebbling model (and not the parallel model) in proving our lowerbound below, our results are incomparable to these previous lower bound results in the parallel model. Our graph constructions are new, and their tightness in the parallel pebbling model is an open question.

In our construction, we make use of the stacked superconcentrators constructed in [LT82, §4] except that the vertices are connected in some topological order (blowing up our graph by only a constant factor of 6 if we replace all degree 3 nodes with a height 3 pyramid).

Graph Construction 6.14. *Let $C(n, k)$ be a stacked superconcentrator with k layers where C_i is the i-th linear superconcentrator. We create the following edges between nodes. Let* T *be a topological sort order of the vertices in $C(n, k)$. Create edges (v_i, v_{i+1}) where v_i is the vertex immediately preceding v_{i+1} in* T*. Replace all degree* 3 *nodes with pyramids of height* 3*.*

It was proven in [LT82] (Theorem 4.2.6) that given $S \leq \frac{n}{20}$ pebbles, k layers, and n nodes in each linear superconcentrator per layer, the pebbling time, $T(n, k, S)$, of pebbling $C(n, k)$ is lower bounded by:

$$T(n, k, S) = n\Omega\left(\left(\frac{nk}{64S}\right)^k\right).$$

In our construction defined by Def. 6.14, we first let $S = c_1(N \log\log N/\log N)$ (for some constant c_1), $n = 20S$, $k = \lfloor N/S \rfloor$, and we get a graph $C(n, k)$ with $\Theta(N)$ vertices. Thus, we obtain the following tradeoff for this graph given S pebbles:

$$T \geq S\Omega\left(\frac{N}{S}\right)^{\Omega(N/S)}$$

for $S \leq c_2\left(\frac{N \log\log N}{\log N}\right)$ for some constant c_2 where $c_2 < c_1$.

Thus, we notice two main characteristics of our graph. If $S \geq c_1\left(\frac{N \log\log N}{\log N}\right)$, then the time it takes to pebble the graph is $O(N)$ since the width of the graph is $\Theta\left(\frac{N \log\log N}{\log N}\right)$. Second, if $S \leq c_2\left(\frac{N \log\log N}{\log N}\right)$ then S pebbles are used to pebble the graph for $\omega(N)$ time by Theorem 4.2.6 of [LT82]. Note that if the tradeoff is sufficiently great, then we achieve our stated lower bound. To prove our stated lower bound, we modify the proof for Theorem 4.2.5 of [LT82] so that we account for CC^α instead of just the time-space tradeoff. Minimizing the equation for tradeoff in terms of $\alpha = 1$ and showing that the cost is greater than the cost of when $S \geq c_1\left(\frac{N \log\log N}{\log N}\right)$ and the cumulative complexity for when $S \geq c_1\left(\frac{N \log\log N}{\log N}\right)$ is $\Theta\left(\frac{N^2 \log\log N}{\log N}\right)$ then provides us with the lower bound we want.

We use the same notation as that used in the proof of Theorem 4.2.5 in [LT82]. Let n be the number of outputs of the superconcentrator $C(n, k)$ and k be the number of copies of the

[20]Although our construction matches asymptotically the best lower bound construction in the pROM (see the footnote for Lemma 6.22).

linear superconcentrators (number of levels in the stack of superconcentrators) in $C(n,k)$. We number the parts of $C(n,k)$ similarly to how they are numbered in the proof of Theorem 4.2.5, let C_i be the i-th copy of the linear superconcentrators that composes $C(n,k)$. We consider the outputs of C_k as numbered in the order in which they are first pebbled. Let z_i be the time that output i (where $1 \leq i \leq n$) is pebbled. Therefore, $z_0 = 0$ and $z_{n+1} = \mathsf{Time}(C(n,k), S)$. Then, let $[z'_i, z''_i]$ be the interval of time starting with the z'_i-th move and ending with the z''_i-th move where $z_{i-1} \leq z'_i \leq z''_i \leq z_i$. Let p_i be the minimum number of pebbles on C_k in the interval $[z_{i-1}, z_i]$ for $1 \leq i \leq n$ and where $p_0 = 0$, $p_{n+1} = 0$, and $p_i \leq S$ for all i in the valid range.

We first note that since we do not remove any vertices or edges (only add edges to the construction to maintain the topological order and to ensure that at most one additional pebble is added to the graph at each time step), all properties of the graph with respect to n as proven in [LT82] still hold (i.e. adding edges does not change the linear superconcentrator properties of the graphs). Hence, we restate some of the key theorems and lemmas in [LT82] that will allow us to prove the lower bound in CC^α when $\alpha = 1$ that we seek.

We restate the definition of a good interval given in [LT82] below:

Definition 6.15 (Good Intervals [LT82])**.** *An interval $[i,j] \subset [1,n]$ is* good *if it fulfills the following three requirements:*

$$p_i \leq \frac{j-i}{2}, \tag{8}$$

$$p_{j+1} \leq \frac{j-i}{2}, \tag{9}$$

$$p_k > \frac{j-i}{8} \text{ for } i < k \leq j. \tag{10}$$

We also restate one key lemma relating to good intervals below:

Lemma 6.16 (Lemma 4.2.3 [LT82])**.** *During the good interval $[i,j]$ at least $n-2S$ different outputs of C_{k-1} are pebbled. Only $S-1-\lfloor\frac{j-i}{8}\rfloor$ pebbles are available to pebble the $n-2S$ different outputs of C_{k-1}.*

We also restate a combinatorial lemma proved in [LT82] that will allow us to prove a recursive relation on CC^α (which will subsequently allow us to provide a bound for our construction).

Lemma 6.17 (Lemma 4.2.4 [LT82])**.** *Let $r \leq n$. We can find a set of disjoint good intervals in $[1,r]$ that covers at least $\frac{r}{4} - S - p_{r+1}$ elements of $[1,r]$.*

Finally, we adapt a theorem based on a simple application of BLBA that provides a (not quite tight enough) lower bound on the time necessary to pebble our constructed graph given S pebbles and provide a proof for our construction defined in Graph Construction 6.14.

Theorem 6.18 (Theorem 4.2.1 [LT82])**.** *In order to pebble all outputs of $C(n,k)$ as defined in Graph Construction 6.14 using S black pebbles, $2 \leq S \leq \frac{n-1}{4}$ (starting with any configuration of pebbles on the graph), we need T placements where*

$$T \geq n\left(\frac{n}{10S}\right)^k.$$

Using these lemmas, we now write our final recursive theorem for the CC^α of our construction.

Theorem 6.19. *Let $CC^\alpha(N, k, S)$ be the CC^α (when $\alpha = 1$) necessary to pebble all the outputs of $C(n, k)$ (recall that the topological sort of the vertices requires that for the last output to be pebbled, all other outputs must be pebbled) with $S \leq \frac{n}{20}$ pebbles. Then,*

$$T(n, 1, S) \geq \frac{n^2}{10S} \tag{11}$$

$$T(n, k, S) \geq \min_{(x_1,\ldots,x_m)\in D_k} \sum_{1\leq i\leq m} T\left(n, k-1, S-1-\left\lfloor \frac{x_i - 1}{8} \right\rfloor\right) \text{ for } k > 1, \tag{12}$$

$$CC^\alpha(N, k, S) \geq \min_{D_1,\ldots,D_k} \sum_{1\leq j\leq k} \sum_{(x_1,\ldots,x_m)\in D_j} \left\lfloor \frac{x_i - 1}{8} \right\rfloor \left(T\left(n, j-1, S-1-\left\lfloor \frac{x_i - 1}{8} \right\rfloor\right)\right) \tag{13}$$

$$\geq \min_{D} \sum_{(x_1,\ldots,x_m)\in D} \left\lfloor \frac{x_i - 1}{8} \right\rfloor T\left(n, k-1, S-1-\left\lfloor \frac{x_i - 1}{8} \right\rfloor\right). \tag{14}$$

where D_i is an index set that contains all the ways in which we can select a large number of good intervals. Specifically,

$$D_i = \left\{(x_1, \ldots, x_m) | m > \frac{n}{64S}, 1 \leq x_i \leq 8S-6 \text{ for } 1 \leq i \leq m, \text{ and } \sum_{1\leq i\leq m} x_i \geq \frac{n}{8}\right\}.$$

Proof. The proof for the expression for $T(n, k, S)$ follows directly from Theorem 4.2.5 in [LT82].

Now we prove the expression for CC$^\alpha$ of $C(n, k)$ for the case when $S \leq n/20$. For each good interval, at least $\left\lfloor \frac{x_i-1}{8} \right\rfloor$ pebbles must remain on C_k while $C_1, \ldots, C_{k-1}$ are pebbled with the remaining $S - 1 - \left\lfloor \frac{i-1}{8} \right\rfloor$ pebbles. Therefore, the CC$^\alpha$ when $\alpha = 1$ of the good period with length x is $\left\lfloor \frac{x_i-1}{8} \right\rfloor T\left(n, k-1, S-1-\left\lfloor \frac{x_i-1}{8} \right\rfloor\right)$. By Lemma 6.17, we have that the total length of the disjoint good intervals is at least $n/8$ (since $p_{r+1} \leq S$ and $n/4 - 2S \geq n/8$). Thus, summing over the CC$^\alpha$ for all good intervals and minimizing over all possible allocations of good intervals gives a lower bound on the CC$^\alpha$ for C_k which is a lowerbound on the CC$^\alpha$ when $\alpha = 1$ of the entire graph. □

Lemma 6.20. *When $S = c_1\left(\frac{N \log\log N}{\log N}\right)$ for some constant c_1, $n = 20S$, $k = \lfloor N/S \rfloor$ and we create a graph according to Graph Construction 6.14, $C(n, k)$ with $\Theta(N)$ vertices,*

$$CC^\alpha(N, k, S) \geq \min_{D} \sum_{(x_1,\ldots,x_m)\in D} \left\lfloor \frac{x_i - 1}{8} \right\rfloor \left(20S\left(\frac{20S(\lfloor N/S \rfloor - 1)}{c\left(S-1-\left\lfloor \frac{x_i-1}{8} \right\rfloor\right)}\right)^{\lfloor N/S \rfloor - 1}\right) \tag{15}$$

for $S \leq c_2\left(\frac{N \log\log N}{\log N}\right)$ for some constants c (specified in the proof) and $c_2 < c_1$.

Proof. We know from [LT82] that the expression for $T(n, k, S)$ is lower bounded by $T(n, k, S) \geq n\left(\frac{nk}{cS}\right)^k$ for some constant $c \geq 10$. Therefore, we can substitute this expression into our Eq. 14 to obtain the following expression:

$$\text{CC}^\alpha(N, k, S) \geq \min_{D} \sum_{(x_1,\ldots,x_m)\in D} \left\lfloor \frac{x_i - 1}{8} \right\rfloor \left(n\left(\frac{n(k-1)}{c(S-1-\left\lfloor \frac{x_i-1}{8} \right\rfloor)}\right)^{k-1}\right).$$

Substituting our values as stated above then gives

$$\mathrm{CC}^{\alpha}(N,k,S) \geq \min_{D} \sum_{(x_1,\ldots,x_m)\in D} \left\lfloor \frac{x_i - 1}{8} \right\rfloor \left(20S \left(\frac{20S(\lfloor N/S \rfloor - 1)}{c\left(S - 1 - \lfloor \frac{x_i - 1}{8} \rfloor\right)} \right)^{\lfloor N/S \rfloor - 1} \right) \tag{16}$$

for some number of pebbles used that is less than $n/20$; or in other words, for some constant c_2, $S \leq c_2 \left(\frac{N \log\log N}{\log N}\right)$ where we determine the exact values of c_1 and c_2 later on (since the exact values of c_1 and c_2 also depend on the types of linear superconcentrators used in each of the k layers of our construction). □

Lemma 6.21. *Given $S \leq c_2 \left(\frac{N \log\log N}{\log N}\right)$ for some constant c_2 where $c_2 \left(\frac{N \log\log N}{\log N}\right) < n/20$,*

$$CC^{\alpha}(N,k,S) \geq \frac{c_2}{8} \left(\frac{N \log\log N}{\log N}\right) \left(20S \left(\frac{20S\left(\lfloor \frac{N}{S} \rfloor - 1\right)}{c(S-1)} \right)^{\lfloor \frac{N}{S} \rfloor - 1} \right). \tag{17}$$

Proof. We assume for the sake of contradiction that there exists a closed formed lowerbound for the equation where some $x_i > 1$. Suppose there exists some good period with length $x_i > 1$, then the term

$$\frac{x_i}{8} \left(20S \left(\frac{20S\left(\lfloor \frac{N}{S} \rfloor - 1\right)}{c(S - 1 - \lfloor \frac{x_i - 1}{8} \rfloor)} \right)^{\lfloor \frac{N}{S} \rfloor - 1} \right)$$

is in the summation of the calculation of $\mathrm{CC}^{\alpha}(N,k,S)$ (see Eq. 16). We can replace the term with the following:

$$x_i \left(\frac{1}{8} \left(20S \left(\frac{20S\left(\lfloor \frac{N}{S} \rfloor - 1\right)}{c(S-1)} \right)^{\lfloor \frac{N}{S} \rfloor - 1} \right) \right)$$

which results in a smaller $\mathrm{CC}^{\alpha}(N,k,S)$ a contradiction, therefore no values of x_i are greater than 1 and the closed form lower bound is that as stated in Eq. 17. □

Lemma 6.22. *Given $S \leq c_2 \left(\frac{N \log\log N}{\log N}\right)$ for some constant c_2 where $c_2 \left(\frac{N \log\log N}{\log N}\right) < n/20$, CC^{α} when $\alpha = 1$ is $\omega\left(\frac{N^2 \log\log N}{\log N}\right)$.*[21]

Proof. From Lemma 6.21, the CC^{α} when less than $c_2 \left(\frac{N \log\log N}{\log N}\right)$ pebbles are used is lower bounded by the closed form expression,

$$\mathrm{CC}^{\alpha}(N,k,S) \geq \frac{c_2}{64} \left(\frac{N \log\log N}{\log N}\right) \left(20S \left(\frac{20S\left(\lfloor \frac{N}{S} \rfloor - 1\right)}{c(S-1)} \right)^{\lfloor \frac{N}{S} \rfloor - 1} \right). \tag{18}$$

[21]We can show for this case that CC^{α} is $\omega\left(\frac{N^2}{\log N}\right)$ in the parallel random oracle case since the runtime in Eq. 18 can be improved by at most a factor of $\frac{1}{S}$.

We know that the lower bound given in Eq. 18 is $\Theta\left(\frac{N\log\log N}{\log N}\left(S\left(\frac{N}{S}\right)^{\frac{N}{S}-1}\right)\right)$.

Given $S \leq \frac{N\log\log N}{\log N}$ pebbles, we now prove that the CC^α of our construction for $\alpha = 1$ is $\omega\left(\frac{N^2\log\log N}{\log N}\right)$. We know that $S\left(\frac{N}{S}\right)^{\frac{N}{S}-1} = \omega(N)$ for all $S \leq c_2\left(\frac{N\log\log N}{\log N}\right)$. Therefore, $\mathrm{CC}^\alpha(N,k,S) = \omega\left(\frac{N^2\log\log N}{\log N}\right)$. $\square$

Theorem 6.23. *Given* $S > c_2\left(\frac{N\log\log N}{\log N}\right)$*,* CC^α *when* $\alpha = 1$ *is* $\Omega\left(\frac{N^2\log\log N}{\log N}\right)$*. Therefore,* $CC^\alpha(G) = \Theta\left(\frac{N^2\log\log N}{\log N}\right)$ *in the sequential*[22] *pebbling model where* G *is given by our Graph Construction 6.14 above.*

Proof. Let S be large enough that a single linear superconcentrator with n output nodes can be pebbled in almost linear time. In this case, we use the simple BLBA argument presented in Theorem 4.2.1 of [LT82] to prove that in this case, $\mathrm{CC}^\alpha(N,k,S) = \Omega\left(\frac{N^2\log\log N}{\log N}\right)$ since each C_i in the construction of $C(n,k)$ as defined in Graph Construction 6.14 along with the edges joining C_{k-1} with C_k is an n-superconcentrator.

The BLBA theorem as proven in [LT82] proves a tradeoff in time with respect to the number of pebbles in the starting and ending configuration of the graph. Let S_a be the starting number of pebbles on the graph and S_b be the ending number of pebbles on the graph. Suppose that $S_b = 0$ for the sake of lowerbounding our cumulative complexity. Then $c_2\left(\frac{N\log\log N}{\log N}\right) < \min_{1\leq i\leq k}\left(S_a^i\right) \leq S$ by our theorem statement where S_a^i is the starting pebble configuration for level i. Suppose that $S_a^{c_i} \leq c_2\left(\frac{N\log\log N}{\log N}\right)$ for L levels (i.e. for some set of levels in $[c_1,\dots,c_L]$), then $\mathrm{CC}^\alpha(n,i,S)$ is given by Lemma 6.22 for the L values. Using Lemma 6.22, we see that in order for the bound from Lemma 6.22 to not hold, we must have $L = o(N/S)$. But, then, $N/S - o(N/S) = \Theta(N/S)$ layers are pebbled with $S_a^i > c_2\left(\frac{N\log\log N}{\log N}\right)$ pebbles. Therefore, we achieve the same asymptotic bound by considering $c_2\left(\frac{N\log\log N}{\log N}\right) < \min_{1\leq i\leq k}\left(S_a^i\right) \leq S$.

Thus, by BLBA, we know that

$$T(n,1,S) \geq \max\left(1, \frac{n-2S}{2S+1}\right) \tag{19}$$

$$T(n,i,S) \geq n\left(\max\left(1,\frac{n}{10S}\right)\right)^i \tag{20}$$

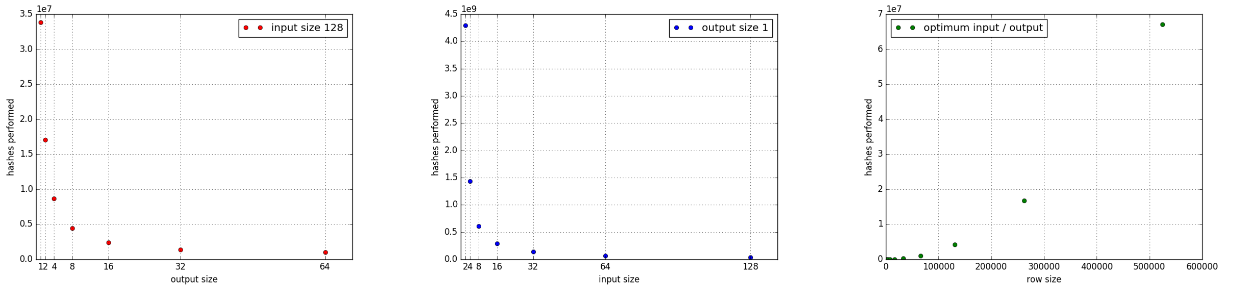
$$\mathrm{CC}^\alpha(N,k,S) \geq \sum_{1\leq i\leq k:S_a^i} S_a^i T(n,i-1,S-S_a^i)\max\left(1,\left(\frac{n-2S_a^i}{2S_a^i+1}\right)\right) \tag{21}$$

$$\geq n\min_{1\leq i\leq k:S_a^i}\left(S_a^i\right)\max\left(1,\frac{n-2S}{2S+1}\right)(k-1) \tag{22}$$

We can simplify in the last step since $T(n,i-1,S-S_a^i) \geq n$ for all $1 \leq i \leq k$. Furthermore, by our argument above, we know that $\min_{1\leq i\leq k:S_a^i}\left(S_a^i\right) = \Theta(S)$.

When $n = c_1\left(\frac{N\log\log N}{\log N}\right)$, $S > c_2\left(\frac{N\log\log N}{\log N}\right)$, and $k = \frac{\log N}{\log\log N}$, then Eq. 22 simplifies to $\Omega\left(\frac{N^2\log\log N}{\log N}\right)$ for some predefined c_2 and c_1. Otherwise, the time of pebbling is N using

[22]Erratum: An earlier version of this paper stated the theorem for general pebbling strategies, not just sequential ones. The proof herein is unchanged from that earlier version, and proves the theorem only for sequential strategies.



(a) Runtime vs. output size, 128 B input, 65 KB row (b) Runtime vs. input size, 1 B output, 65 KB row (c) Runtime vs. row size, 64 B output, 128 B input

Figure 5: Evaluation of cylinder implementation

$c_1\left(\frac{N\log\log N}{\log N}\right)$ pebbles resulting in CC^α when $\alpha=1$ to be $\Theta\left(\frac{N\log\log N}{\log N}\right)$. $\square$

Case of $\alpha = 2$ We briefly note that the above construction does not asymptotically achieve tightness for $\alpha=2$ by our current analysis. This is due to the fact that when $\alpha=2$, Lemma 6.22 no longer holds due to the fact that $\left(\frac{N\log\log N}{\log N}\right)\cdot\left(\frac{\log N}{\log\log N}\right)^{\frac{\log N}{\log\log N}} = o(N^2)$.

Open Question. *Does there exist a bounded in-degree graph family that has* CC^α *for* $\alpha \geq 2$ *that meets the upper bound?*

7 Cylinder-based SHF implementation

We implemented a prototype our *cylinder* construction defined in Def. 5.4. We choose to implement this construction because it is simplest of the constructions we present for $\mathcal{H}_1$, yet achieves memory and time bounds comparable to our more complicated construction. Our implementation seeks to give a preliminary demonstration of practical feasibility of the cylinder construction for certain parameter ranges; it is not a detailed evaluation of optimized performance.

In implementing the pebbling construction, we seek to minimize the runtime of $\mathcal{H}_1$ while maximizing its output size. This leads to some interesting tradeoffs as well as an observation about static-memory-hardness and the random oracle model in general.

Overview of implementation First, we map an entire row of labels (i.e., labels in a particular layer of our construction) in our cylinder construction defined in Definition 5.4 to an array of bits in memory of length l. We implement a serialized pebbling algorithm by iteratively reading n bits, starting at offset f, applying a hash function to the read bits, writing the n-bit output from the hash function at offset f, and finally incrementing f by additive n bits for the next round. This process is repeated until the end of the string, which constitutes one row of the cylinder construction. This procedure of processing the rows of the cylinder is repeated once for every row of the cylinder DAG.

Parameters Our configurable parameters are: the total size of the array l, the input size i and the output size n of the hash function. Other parameters depend on these as follows:

- The *label size* is n which is also the output length of the hash function. Every hash produces one label. The number of labels per row is then $\frac{l}{n}$. l should be a multiple of n so that there are no partial labels at the end of the array.
- The *indegree* of the wraparound pyramid is $\frac{i}{n}$. Here as well, i should be a multiple of n so that the degree is an integer and this maps cleanly to the pebbling model when we consider ingree

$2n$ (ie constant indegree 2 in the pebbling model).

- The *height* needed for the wraparound pyramid is then the array size divided by the difference between input and output sizes, or $\frac{l}{i-n}$. The input size must be greater than the output size for the height to be defined. This corresponds with the requirement that the degree must be at least 2 for the pyramid construction to provide meaningful guarantees.

Instantiating the random oracle We used blake2b, a fast and well-known hash function. Blake2b has an internal state size of 1024 bits, so we were able to set i to 1024 bits while keeping the memory hardness. n was set to 512 bits, giving a 2-degree pebbling graph. Decreasing either i or n would lead to inefficient use of the function. It would seem that hash functions with a larger internal state size, capable of supporting a larger i would be faster for this usage, but it is not as clear as larger state sizes may correlate with slower evaluation of a single hash.

We measured the time taken by using a single core of an AMD Ryzen 7 1700 processor. The single threaded code was able to perform approximately 300 million hash operations per second on 1024 bit inputs. This rate could be increased by using multiple cores, but the 300 million hashes per second rate can be used with the above figures to see how many hash operations are being performed at the different settings.

7.1 Remarks on implementation and musings on random oracles in practice

Reducing number of hashes The runtime of evaluating $\mathcal{H}_1$ is determined by the number of hash transformations called as very little other computation is done. The number of hashes per row is $\frac{l}{n}$, and the number of rows is $\frac{l}{i-n}$, giving a total number of hash calls as

$$\left(\frac{l}{n}\right)\left(\frac{l}{i-n}\right) = \frac{l^2}{ni-n^2} \ .$$

l^2 indicates the expected time requirement proportional to the square of the output size. To optimize the time for a given l, we look at the denominator, $ni - n^2$, noting that $i > n$, keeping this positive. To reduce the time taken, increase the input size i. Graphically, this makes sense as descending from the top of the wraparound pyramid, the higher degree will quickly cover the entire width of a row. However, in practice we cannot increase i and maintain the memory-hard properties: this is an interesting divergence between the random oracle model and real-world hash functions.

Data busses to the random oracle One aspect which is rarely discussed in the random oracle model is the exact process by which one makes a call to the oracle. Does the query need to be sent to the oracle via a parallel bus, all bits at once, or is the query sent via a serial bus, one bit at a time? If serially, can we send some of the bits, then wait a while, and send the rest? We are not aware of literature dealing with these mechanics of data transmission to and from the oracle; however, in our case it is quite relevant. If serial transmission is allowed, i can be made arbitrarily large without needing to store the whole row of the wraparound pyramid in memory. For each bit of a label, as soon as it is computed it can be sent to all the oracles using that bit as an input, and promptly forgotten; the oracles act as a memory cache. The memory-hardness proofs implicity assume an oracle model where the entire query is handed over simultaneously to the oracle, and as such, any query to the oracle must exist in its entirety in memory before the query is made.

In practice, real-world hash functions resemble a serial-bus oracle much more closely than a parallel bus oracle. Whether we're referring to Merkle-Damgård [], Sponge construction [], or other methods, today's widely used hash functions are built out of fixed length one-way functions. The internal state of a hash function can thus act as a data cache for the purposes of the pebbling graph. For a high-degree node, the left predecessors can be fed to the hash function and forgotten before the right predecessors are known. Since the internal state of the hash function has a fixed size, this defeats the memory hardness promised by the pebbling construction.

Data-dependence and cache timing attacks We implement a data-dependent memory access pattern for $\mathcal{H}_2$. Other papers (e.g., Catena [LW15] and Balloon Hash [BCGS16]) have identified security vulnerabilities due to data-dependent memory access patterns which can leak information about the password to an attacker with incomplete access to the physical system evaluating the password hashing function. These attacks occur because of the variable time taken to evaluate the function based on the input data, primarily due to the automatic caching of data inside the CPU.

We believe that our $\mathcal{H}_2$ function, while implementing a data-dependent memory access pattern, is likely much more resistant to cache timing attacks than the examples mentioned above, based on the following analysis. In practical usage, the output of $\mathcal{H}_1$ will be very large with respect to the number of queries $\mathcal{H}_2$ performs on the data; for any single evaluation of $\mathcal{H}_2$, nearly all of the $\mathcal{H}_1$ data will go unread. Because of this sparse access, *data will be read once and not used again before being evicted from the cache.* The probability that an input to $\mathcal{H}_2$ results in a collision, and multiple reads from the same memory region, is thus modeled by $(\llbracket\Lambda\rrbracket)^{-Q}$ where Λ is the size of the output of $\mathcal{H}_1$, and Q is the number of oracle calls made by $\mathcal{H}_2$.

Inputs resulting in cache hits should be rare, and knowledge of a cache hit during $\mathcal{H}_2$ evaluation give a bounded advantage to the attacker expressed by

$$\frac{\text{total number of access patterns with } n \text{ collisions}}{\text{total number of zero-collision access patterns}} .$$

However, we note that this could still be significant advantage in practice because attackers do not need to perform memory lookups into the set of $\mathcal{H}_1$ outputs in order to detect collisions. That is, an attacker still has to perform lots of hashes, but their memory requirement could go down significantly.

On memory allocation In order to implement the wraparound pyramid in the efficient way described above, memory usage needs to be slightly greater than that stated in theoretical model, due to necessary memory allocations in the hardware. Namely, the leftmost bits of the array need to be copied and appended to the right side, so that the lower level input values are available to the final hash interations which consume the wraparound inputs. This increases the memory needed by $i - n$.

Acknowledgements

We are grateful to Jeremiah Blocki, Krzysztof Pietrzak, and Joël Alwen for valuable feedback on earlier versions of this paper. We thank Ling Ren for helpful technical discussions. We also thank Erik D. Demaine and Shafi Goldwasser for their advice on this paper. Finally, we thank our anonymous reviewers for insightful comments.

Sunoo's research is supported by NSF MACS (CNS-1413920), DARPA IBM (W911NF-15-C-0236), SIMONS Investigator Award Agreement Dated June 5th, 2012, and the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370.

References

[AAC$^+$17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. 10625:357–379, 2017.

[AB16] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 241–271. Springer, 2016.

[ABH17] Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In *CCS*, pages 1001–1017. ACM, 2017.

[ABP17a] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In *EUROCRYPT (3)*, volume 10212 of *Lecture Notes in Computer Science*, pages 3–32, 2017.

[ABP17b] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. *CoRR*, abs/1705.05313, 2017.

[ACK$^+$16] Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 358–387. Springer, 2016.

[ACP$^+$16] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. *IACR Cryptology ePrint Archive*, 2016:989, 2016.

[ADN$^+$10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2010.

[AdRNV17] Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. Cumulative space in black-white pebbling and resolution. In *ITCS*, volume 67 of *LIPIcs*, pages 38:1–38:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In Kaoru Kurosawa, editor, *Information Theoretic Security, 4th International Conference, ICITS 2009, Shizuoka, Japan, December 3-6, 2009. Revised Selected Papers*, volume 5973 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 595–603. ACM, 2015.

[AT17] Joël Alwen and Björn Tackmann. Moderately hard functions: Definition, instantiations, and applications. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15,* *2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 493–526. Springer, 2017.

[BCGS16] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. *Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks*, pages 220–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[BDPA08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 181–197, 2008.

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

[BK15] Alex Biryukov and Dmitry Khovratovich. Tradeoff cryptanalysis of memory-hard functions. In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 633–657. Springer, 2015.

[BKR16] Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-key symmetric encryption: Resisting key exfiltration. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 373–402. Springer, 2016.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73. ACM, 1993.

[BZ16] Jeremiah Blocki and Samson Zhou. On the computational complexity of minimal cumulative cost graph pebbling. *CoRR*, abs/1609.04449, 2016.

[BZ17] Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of argon2i. In *TCC (1)*, volume 10677 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2017.

[CB02] Richard Clayton and Mike Bond. Experience using a low-cost FPGA design to crack DES keys. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 579–592. Springer, 2002.

[CDD$^+$07] David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard J. Lipton, and Shabsi Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 479–498. Springer, 2007.

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

[CLW06] Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In Halevi and Rabin [HR06], pages 225–244.

[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.

[CS74] Stephen Cook and Ravi Sethi. Storage requirements for deterministic / polynomial time recognizable languages. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 33–39, New York, NY, USA, 1974. ACM.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. *Proofs of Space*, pages 585–605. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[DKW10] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable and uncomputable functions. *IACR Cryptology ePrint Archive*, 2010:541, 2010.

[DKW11] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 125–143. Springer, 2011.

[DL17] Erik D. Demaine and Quanquan C. Liu. Inapproximability of the standard pebble game and hard to pebble graphs. In *Proceedings of the 16th International Symposium on Algorithms and Data Structures, WADS '17*, volume 10389 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2017.

[Dzi06] Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In Halevi and Rabin [HR06], pages 207–224.

[ER61] Paul Erdős and Alfréd Rényi. On a classical problem of probability theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 6:215–220, 1961.

[FLW13] Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A memory-consuming password scrambler. *IACR Cryptology ePrint Archive*, 2013:525, 2013.

[GLT80] John R. Gilbert, Thomas Lengauer, and Robert Endre Tarjan. The pebbling problem is complete in polynomial space. volume 9, pages 513–524, 1980.

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

[HR06] Shai Halevi and Tal Rabin, editors. *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*. Springer, 2006.

[JWK81] Hong Jia-Wei and H. T. Kung. I/o complexity: The red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '81, pages 326–333, 1981.

[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.

[LW15] Stefan Lucks and Jakob Wenzel. Catena variants - different instantiations for an extremely flexible password-hashing framework. In *Technology and Practice of Passwords - 9th International Conference, PASSWORDS 2015, Cambridge, UK, December 7-9, 2015, Proceedings*, pages 95–119, 2015.

[Mer79] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford, CA, USA, 1979. AAI8001972.

[Nor12] Jakob Nordström. On the relative strength of pebbling and resolution. *ACM Trans. Comput. Log.*, 13(2):16:1–16:43, 2012.

[Nor15] Jakob Nordstrom. New wine into old wineskins: A survey of some pebbling classics with supplemental results. 2015.

[Per09] Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009. Presented at BSDCan 2009. Available online at: `http://www.tarsnap.com/scrypt/scrypt.pdf`.

[PH70] Michael S. Paterson and Carl E. Hewitt. Record of the project mac conference on concurrent systems and parallel computation. chapter Comparative Schematology, pages 119–127. ACM, New York, NY, USA, 1970.

[Pip82] Nicholas Pippenger. Advances in pebbling (preliminary version). In *ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 407–417. Springer, 1982.

[Pot17] Aaron Potechin. Bounds on monotone switching networks for directed connectivity. *J. ACM*, 64(4):29:1–29:48, 2017.

[RD17] Ling Ren and Srinivas Devadas. Bandwidth hard functions for ASIC resistance. In *TCC (1)*, volume 10677 of *Lecture Notes in Computer Science*, pages 466–492. Springer, 2017.

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

[SS79a] John E. Savage and Sowmitri Swamy. Space-time tradeoffs for oblivious integer multiplication. In Hermann A. Maurer, editor, *Automata, Languages and Programming*, pages 498–504, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

[SS79b] Sowmitri Swamy and John E. Savage. Space-time tradeoffs for linear recursion. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '79, pages 135–142, New York, NY, USA, 1979. ACM.

[Tom81] Martin Tompa. Corrigendum: Time-space tradeoffs for computing functions, using connectivity properties of their circuits. *J. Comput. Syst. Sci.*, 23(1):106, 1981.

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, pages 162–176, 1977.

Algorithm 2 $\mathcal{H}_2^{q'}$

```
On input $(1^\kappa, x)$ and given oracle access to $\mathsf{Seek}_R$ (where $R$ is the string outputted by $\mathcal{H}_1$):
 1. Let $[\![R]\!] = |R|/w$ be the length of $R$ in words.
 2. Query the random oracle to obtain $\rho_0 = \mathcal{O}(x)$ and $\rho_1 = \mathcal{O}(x+1)$.
 3. Use $\rho_0$ to sample randomly $\iota_1, \ldots, \iota_{q'} \in [[\![R]\!]]$.
 4. Query the $\mathsf{Seek}_R$ oracle to obtain $\{y'_i = \mathsf{Seek}_R(\iota_i)\}_{i\in[q']}$.
 5. Output $(y'_1 || \ldots || y'_{q'}) \oplus \rho_1$.
```

A Details of SHF construction with short labels

Definition A.1 (q''-labeling)**.** *Let $G = (V, E)$ be a DAG with maximum in-degree δ, let $\mathfrak{L}$ be an arbitrary "label set," and define $\mathbb{O}(\delta, \mathfrak{L}) = \left(V \times \bigcup_{\delta'=1}^{\delta} \mathfrak{L}^{\delta'} \to \mathfrak{L}\right)$. Let $\mathcal{O}|_{q''}$ be the function that outputs the first q' bits of the output of $\mathcal{O}$. For any function $\mathcal{O} \in \mathbb{O}(\delta, \mathfrak{L})$ and any label $\zeta \in \mathfrak{L}$, the $(\mathcal{O}, \zeta, q'')$-labeling of G is a mapping $\mathsf{label}_{\mathcal{O},\zeta} : V \to \mathfrak{L}$ defined recursively as follows.*[23]

$$\mathsf{label}_{\mathcal{O},\zeta}(v) = \begin{cases} \mathcal{O}|_{q''}(v, \zeta) & \textit{if } \mathsf{indeg}(v) = 0 \\ \mathcal{O}|_{q''}(v, \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{pred}(v))) & \textit{if } \mathsf{indeg}(v) > 0 \end{cases} .$$

Then, we define our family of random oracle functions defined from our hard to pebble graph family constructions.

Definition A.2 (q''-graph function family)**.** *Let $n = n(\kappa)$ and let $\mathbb{G}_\delta = \{G_{n,\delta} = (V_n, E_n)\}_{\kappa\in\mathbb{N}}$ be a graph family. We write $\mathbb{O}_{\delta,\kappa}$ to denote the set $\mathbb{O}(\delta, \{0,1\}^\kappa)$ as defined in Definition A.1. The* q''-graph function family *of $\mathbb{G}$ is the family of oracle functions $\mathcal{F}_{\mathbb{G}}^{q''} = \{\mathfrak{f}_G\}_{\kappa\in\mathbb{N}}$ where $\mathfrak{f}_G = \{f_G^{\mathcal{O}} : \{0,1\}^\kappa \to (\{0,1\}^\kappa)^z\}_{\mathcal{O}\in\mathbb{O}_{\delta,\kappa}}$ and $z = z(\kappa)$ is the number of sink nodes in G. The output of $f_G^{\mathcal{O}}$ on input label $\zeta \in \{0,1\}^\kappa$ is defined to be*

$$f_G^{\mathcal{O}}(\zeta) = \mathsf{label}_{\mathcal{O},\zeta}(\mathsf{sink}(G)) ,$$

where $\mathsf{sink}(G)$ is the set of sink nodes of G.

B Regular and normal pebbling strategies

Here, we restate three theorems and prove briefly their equivalent formulation in the parallel model for the parallel model adapted from theorems in [GLT80, DL17] proven in the sequential model.

We first restate the definitions for normal and regular strategies:

Definition B.1 (Frugal Strategy [GLT80])**.** *Given a DAG $G = (V, E)$, a* frugal strategy *is a pebbling strategy with no unnecessary placements. In particular, the following are true of any frugal pebbling strategy:*

1. *At all times after the first placement on a vertex v, some path from v to the goal vertex contains a pebble.*
2. *At all times after the last placement on a vertex v, all paths from v to the goal vertex contain a pebble.*

[23]We abuse notation slightly and also invoke $\mathsf{label}_{\mathcal{O},\zeta}$ on *sets* of vertices, in which case the output is defined to be a tuple containing the labels of all the input vertices, arranged in lexicographic order of vertices.

3. *The number of placements on a nongoal vertex is bounded by the total number of placements on its successors.*

Definition B.2 (Normal Strategy [GLT80])**.** *A* normal strategy *is a standard pebbling strategy that is frugal and it pebbles each pyramid P in G as follows: after the first pebble is placed on P, no placement or removal of pebbles occurs outside P until the apex of P is pebbled and all other pebbles are removed from P. No new placement occurs on P until after the pebble on the apex of P is removed.*

Theorem B.3 (Normal Strategy Conversion [GLT80])**.** *If the goal vertex is not inside a pyramid, any standard pebbling strategy can be transformed into a normal pebbling strategy without increasing the number of pebbles used in both the sequential and parallel pebbling models.*

Proof. The proof of this statement in the sequential model is given in [GLT80]. We now prove this statement in the parallel model. By our proof of Lemma 5.1, any sequential strategy can be simulated trivially by a parallel strategy; therefore, if any pebbling strategy can be transformed into a sequential normal pebbling strategy, then any pebbling strategy can be transformed into a parallel normal pebbling strategy. □

We now define regular pebbling strategies:

Definition B.4 (Regular Strategy [DL17])**.** *Given a DAG $G = (V, E)$, a* regular strategy *is a standard pebbling strategy that is frugal and after the first pebble is placed on any road graph $R_w \in G$, no placements of pebbles occurs outisde R_w until the set of desired outputs of R_w all contain pebbles and all other pebbles are removed from R_w.*

By the same argument as given for the proof of Theorem B.3, we can prove the equivalent for parallel regular pebbling strategies.

Theorem B.5 (Regular Strategy Conversion [DL17])**.** *Given a DAG $G = (V, E)$, if each input, $i_j \in \{i_1, \ldots, i_w\}$, to a road graph has at most 1 predecessor, any standard pebbling strategy that pebbles a set of desired outputs, $O \subseteq \{o_1, \ldots, o_w\}$, at the same tiime can be transformed into a* regular strategy *without increasing the number of pebbles used.*

In addition, we prove this stronger theorem about the pebbling space complexity of pyramid graphs below than the theorems provided in [GLT80, Nor15] that will be useful for determining the pebbling space complexity of pyramids in the magic pebble game.

Theorem B.6. *Given a pyramid graph Π_h with h levels where level 1 has h nodes and level h has 1 node. Given S pebbles and if all S pebbles are placed on level i of the pyramid and $S < h + 1 - i$, then the apex of the pyramid cannot be pebbled using the rules of the standard pebble game.*

Proof. Given $S < h + 1 - i$ pebbles on the i-th layer of a height h pyramid, we know that the i-th level of the pyramid forms a height $h + 1 - i$ height pyramid with the apex. Thus, by the pebbling space complexity of pyramids, $h + 1 - i$ pebbles are necessary on level $h + 1 - i$ in order to pebble the apex. □